Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Zadel

Vizualizacija odprtih demografskih podatkov

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Alenka Kavčič

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Vizualizacija odprtih demografskih podatkov

Tematika naloge:

V zadnjih letih je v skladu z direktivo EU tudi v Sloveniji vedno več prosto dostopnih javnih podatkov. Odprti podatki pa so, čeprav množično dostopni, v obliki surovih številčnih podatkov, ki so zelo suhoparni, nepregledni in težko razumljivi, uporabnikom pa ne omogočajo širšega pregleda nad njimi. Za lažje razumevanje je tako nujno potreben primeren prikaz podatkov, najbolje v grafični obliki. V okviru diplomske naloge zasnujte in izdelajte spletno aplikacijo za vizualizacijo odprtih podatkov, pri čemer se osredotočite na demografske podatke Statističnega urada Republike Slovenije. Aplikacija naj uporabniku omogoča izbiro podatkov za prikaz in kriterijev za njihovo primerjavo ter pripravi vizualno predstavitev izbranih podatkov, ki omogoča hiter in enostaven pregled podatkov in njihovo lažje razumevanje. Način vizualizacije naj tudi ustrezno prilagodi izbranim podatkom in kriterijem primerjave. Pri zasnovi in realizaciji aplikacije uporabite sodobne spletne tehnologije in orodja.

Za vso pomoč in nasvete pri izdelavi diplomske naloge se zahvaljujem mentorici in asistentu. Za vso podporo in zaupanje čez celoten potek mojega študija se zahvaljujem svoji družini ter boljši polovici.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CSS	cascading style sheet	prekrivni slogi
HTML	hypertext markup language	označevalni jezik za oblikovanje večpredstavnostnih dokumentov
SVG	scalable vector graphic	umerljiva vektorska grafika
JSON	JavaScript object notation	objektna notacija JavaScript
HTTP	hypertext transfer protocol	protokol za prenos hiperteksta

Povzetek

Naslov: Vizualizacija odprtih demografskih podatkov

Avtor: Luka Zadel

Leta 2013 je v veljavo prišla direktiva Evropskega parlamenta in Sveta EU, ki določa spremembe k direktivi iz leta 2003, ki določa nabor pravil za ponovno uporabo informacij javnega sektorja. Določeno je, da so dokumenti, ki jih ustvarijo organi javnega sektorja, zbirka virov, ki lahko koristi gospodarstvu znanja[6]. Slovenija, upoštevajoč, to direktivo nudi dostop do informacij javnega značaja, kasneje odprtih podatkov, preko svojih spletnih strani. Podatkom, ki so nadvse zanimivi in uporabni, pa žal manjka orodje, ki bi pripomoglo k lažji predstavi ter razumevanju.

V okviru diplomskega dela smo izdelali spletno aplikacijo, ki bo rešila najden problem. Aplikacija z uporabo zaledne podatkovne baze ter številnih filtrov, ki jih nastavi uporabnik sam, izriše grafe za prikaz podatkov. Uporabniku je omogočeno, da poljubno nastavi število občin in kriterije, po katerih želi primerjati. Po izbiri opcij aplikacija skozi sistem filtrov ter predelav podatkov določi, kakšen način izrisa je najbolj primeren za predstavitev podatkov. Za izdelavo spletne aplikacije smo uporabili tako imenovani MEAN stack, ki je kratica za tehnologije Mongo DB, Express, AngularJS ter Node.js. Tehnologije smo priredili potrebam ter zadnjim tehnološkim trendom, tako da smo izpustili Express, AngularJS posodobili na Angular2, namesto JavaScript skriptnega jezika pa uporabili TypeScript.

Ključne besede: vizualizacija, odprti podatki, MEAN stack, grafi, demografski podatki, primerjava podatkov

Abstract

Naslov: Visualization of open demographic data

Autor: Luka Zadel

European parliament has in the year of 2013 defined a set of changes to the 2003 directive that defined a set of rules for re-usage of information of public sectors. It is stated, that documents created by the states officials represent a valuable resource of knowledge for the economy[6]. Following that directive the Republic of Slovenia is providing access of its open data on the states websites. While the data was interesting and useful, it lacked the means to make it easier to understand and compare. That is why we decided to make an application that would cover that area. Application, while using the data provided and numerous filters set by the user, can draw graphs and by doing so, help visualize the data. The user can freely decide the number of municipalities and filters to use. After doing so, the application will use the set filters and its programmed logic to decide how to best portray the data. To make the application we used a so-called MEAN stack, which, in short, stands for technologies MongoDB, Express, AngularJS and Node.js. The technologies were adapted to needs and trends, and the final selection was made by dropping Express, updating AngularJS to Angular2 and swapping JavaScrpit with Typescript.

Keywords: visualization, open data, MEAN stack, graphs, demographic data, data comparison

Poglavje 1

Uvod

Državni zbor Republike Slovenije je februarja 2003 sprejel Zakon o dostopu do informacij javnega značaja. S tem je zakonsko uredil ustavno pravico dostopanja do informacij javnega značaja vsakega državljana. Slovenska ustava v 39. členu določa, da ima vsakdo pravico dobiti informacijo javnega značaja, za katero ima po zakonu utemeljen pravni interes, razen v posebnih primerih, ki jih določa zakon. Prav tako se z Zakonom o dostopu do informacij javnega značaja omogoča državljanom ustavna pravica iz 44. člena, ki govori o neposrednem in posrednem sodelovanju državljanov pri upravljanju javnih zadev[7]. Za informacije javnega značaja (v nadaljevanju odprte podatke) je v zakonu zapisano: "Informacija javnega značaja je informacija, ki izvira iz delovnega področja organa, nahaja pa se v obliki dokumenta, zadeve, dosjeja, registra, evidence ali drugega dokumentarnega gradiva, ki ga je organ izdelal sam, v sodelovanju z drugim organom, ali pridobil od drugih oseb"[16].

Odprti demografski podatki služijo v veliko namenov. Predstavljajo izčrpen vir informacij, ki jih lahko uporabimo na primer pri osebnih odločitvah, kot bi bila izbira kraja za nakup nepremičnine, poslovnih odločitvah za izbiro strategije razvoja podjetja, lahko pa tudi za razvoj regij samih.

Za zapis podatkov v podatkovne baze uporabljamo relacijske tabele ali nepovezane dokumente, ki so odlični za shranjevanje podatkov, ne pripomorejo pa k razumevanju le-teh. Izpis je pogosto v obliki tabel, pri katerih

je primerjava podatkov dolgotrajna. Za pomoč pri razumevanju podatkov tako uporabljamo vizualne pripomočke, kot so na primer grafi. Za prikaze različnih razmerij podatkov uporabljamo različne grafe, ki so za to primerni. Uporaba napačnega grafa lahko pripelje do napačnega razumevanja razmerja podatkov, kar je pogosto zlorabljeno v podjetniškem svetu.

V sklopu diplomskega dela smo tako izdelali spletno aplikacijo, ki je sestavljena iz dveh delov. Zaledni del (angl. back end) hrani podatke v podatkovni bazi in jih ob zahtevkih posreduje osprednjemu delu (angl. front end) spletne aplikacije. Katere podatke bo zaledni del posredoval, določi uporabnik skozi uporabniški vmesnik, ki ga predstavlja osprednji del. Uporabniški vmesnik poleg določanja filtrov podatkov skrbi tudi za glavno nalogo celotne aplikacije, vizualizacijo odprtih podatkov.

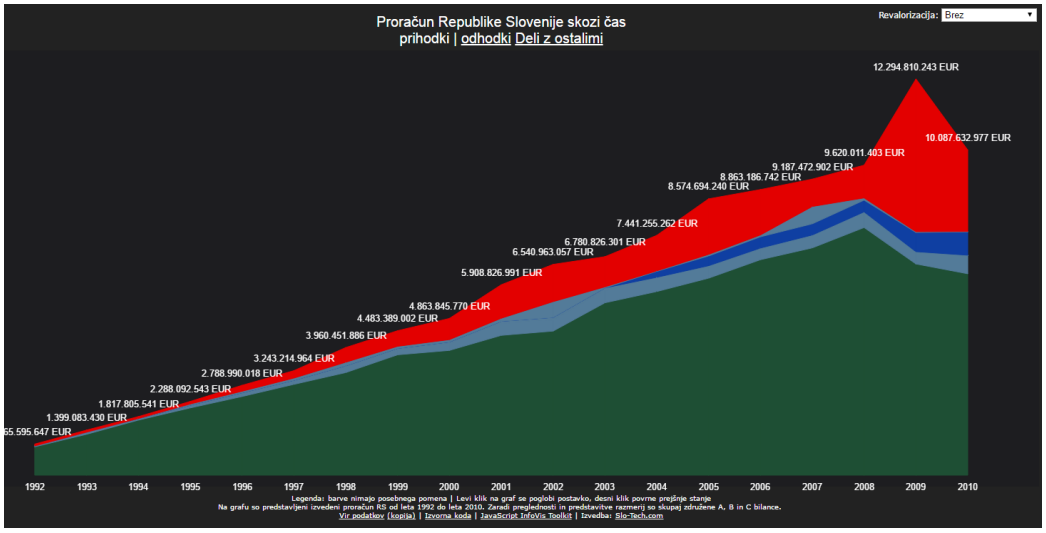
Aplikacija s tako zasnovo prinaša prednost pred statičnimi vizualizacijami podatkov, saj uporabnik lahko določi prikaze, ki jih želi. Statični prikazi podatkov služijo namenu, ampak ne pokrijejo celotnega področja, ki bi ga lahko. Zato je naša rešitev najprimernejši način, saj z dinamičnim prikazom omogoča vizualizacijo vseh kombinacij kriterijev in s tem zagotavlja uporabniku učinkovit prikaz podatkov.

1.1 Pregled obstoječih rešitev za prikaz odprtih podatkov

V Sloveniji obstaja nekaj spletnih aplikacij, ki grafično prikazujejo podatke posameznih področij. Delili jih bomo v podskupine glede na prikaz podatkov. Primeri podskupin so sledeče spletne aplikacije:

- Proračun Republike Slovenije skozi čas je spletna aplikacija, ki uporabniku interaktivno prikazuje različne aspekte proračuna glede na časovno os. Uporabniku dovoljuje revalorizacijo glede na rast bruto družbenega proizvoda ter inflacijo. Ob postavitvi miškinega kazalca uporabnik izve več podatkov o kategoriji, ob kliku pa pridobi pogled, ki predstavlja po-

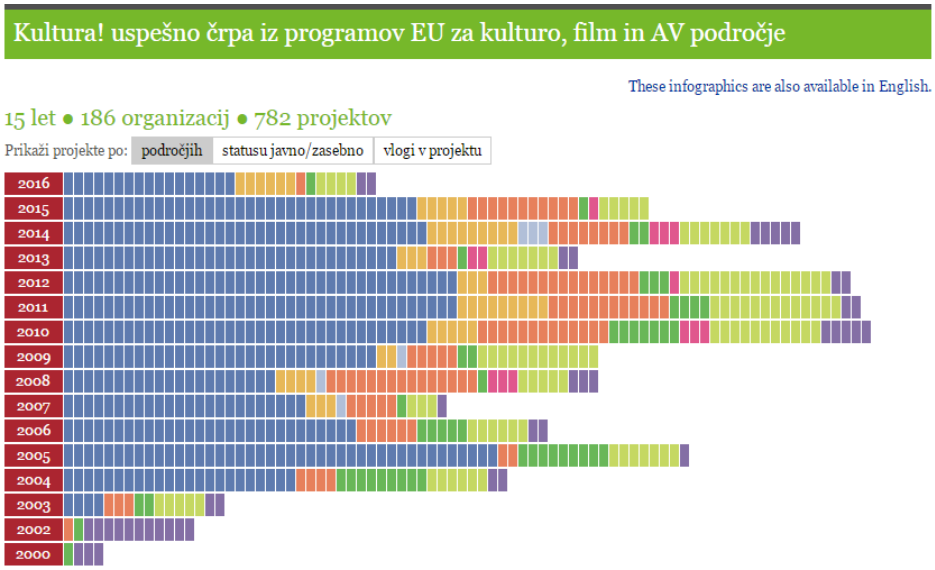
drobnejši opis kategorije. Osnovni prikaz je viden na Sliki 1.1.



Slika 1.1: Proračun republike Slovenije skozi čas. Na sliki je viden prikaz prihodkov skozi čas. Dostopno na: https://static.slo-tech.com/stuff/20letSlovenije/prihodki/nic/, dostopano: 3.1.2017

.

- Grafična in interaktivna vizualizacija podatkov črpanja sredstev iz programov EU omogoča prikaz črpanja sredstev EU za kulturo. Interaktivno prikazuje priliv sredstev skozi čas, razporejeno po področjih, statusu ali pa vlogi v projektu. Aplikacija ob postavitvi miškinega kazalca na posamezen del vizualizacije odebeli robove istega projekta na ostalih časovnih intervalih. Ob kliku uporabnik pridobi dodatne informacije o projektu. Aplikacija ima statičen prikaz podatkov. Aplikacija je na Sliki 1.2.

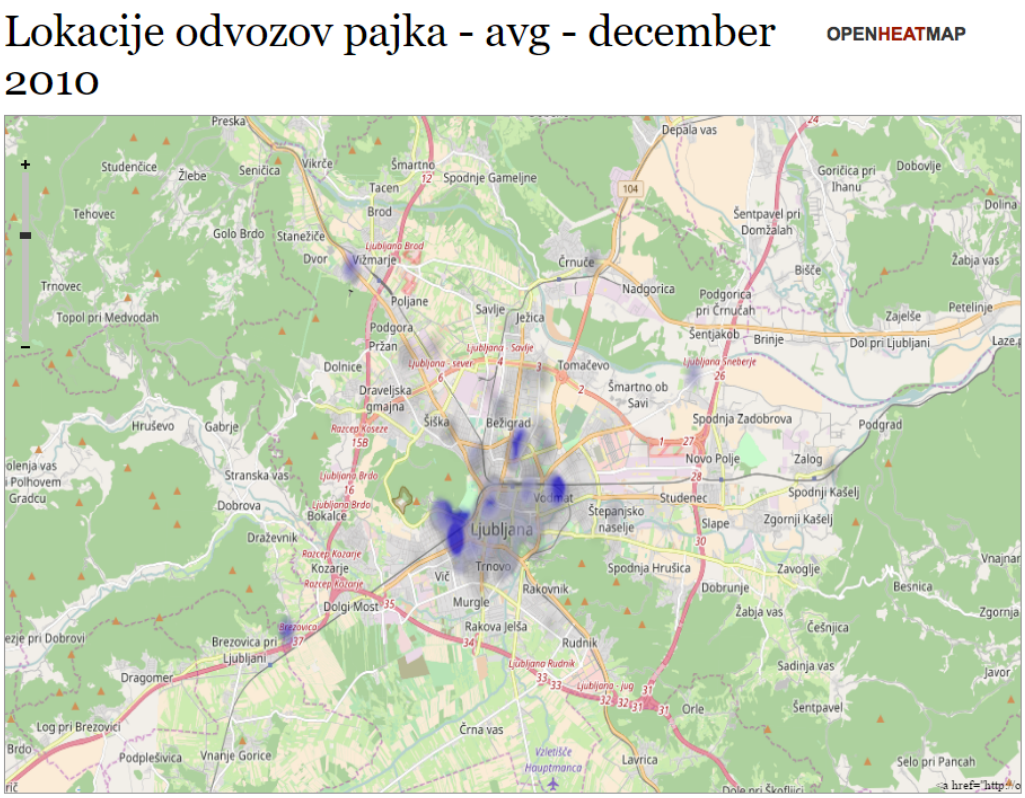


Slika 1.2: Grafična in interaktivna vizualizacija podatkov črpanja sredstev iz programov EU. Na sliki je pogled prikaza črpanja EU sredstev po področjih projektov za kulturo skozi čas. Dostopno na: http://www.culture.si/en/EU_projekti, dostopano: 3.1.2017.

- Aplikacija za prikaz starosti zgradb v Ljubljani. Aplikacija na zemljevid izriše vizualizacijo, ki z barvami ponazori starost zgradb. Uporabniku je podan tudi linijski graf, ki prikaže število zgrajenih zgradb skozi čas. Osnovni primer je na Sliki 1.3. Ob postavitvi miškinega kazalca uporabnik izve točno starost posamezne zgradbe. Poleg pojavnega okna z letnico, aplikacija omogoča le akcije upravljanja z zemljevidom.

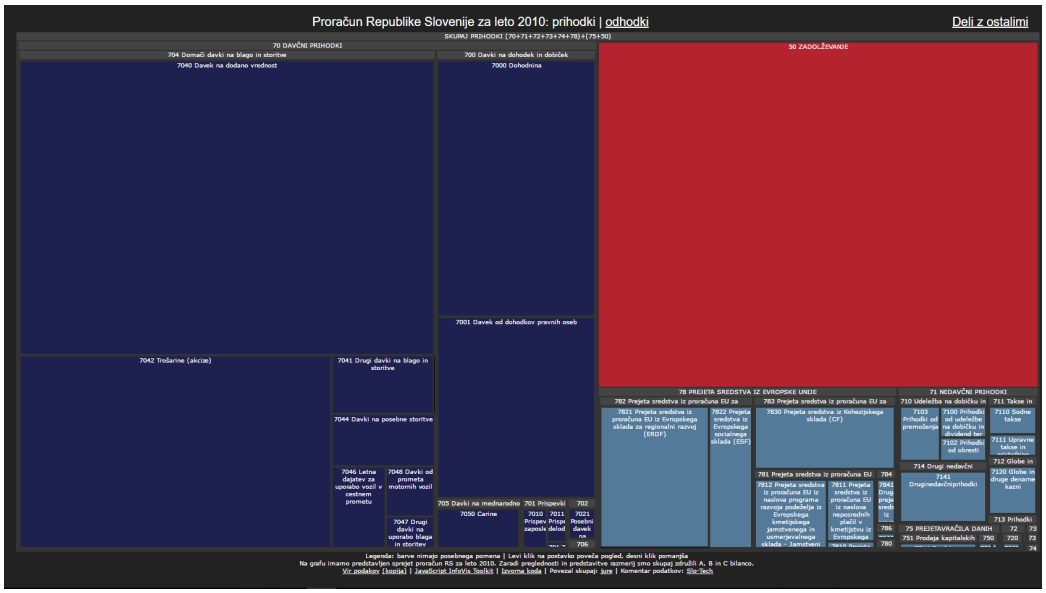
Slika 1.3: Aplikacija za prikaz starosti zgradb v Ljubljani. Slika prikazuje del zemljevida Ljubljane z označenimi zgradbami glede na legendo. V desnem spodnjem delu slike je viden graf količine posameznih kategorij. Dostopno na: http://www.virostatiq.com/data/ljubljana-building-ages/, dostopano: 3.1.2017.

- Lokacije odvozov pajka je aplikacija, ki prikazuje število nepravilno parkiranih avtomobilov, ki jih je občina odstranila. Prikaz gostote je viden na Sliki 1.4. Uporabniku je omogočeno običajno premikanje zemljevida in izpis natančne vrednosti, ki jo vizualizacija predstavlja na določeni točki. Aplikacija je omejena na en kriterij in en nivo vizualizacije.



Slika 1.4: Lokacije odvozov pajka, na sliki je prikaz gostote odstranitve nepravilno parkiranih vozil v Ljubljani za mesec avgust, v letu 2010. Dostopno na: http://www.openheatmap.com/view.html?map=IcecapsNeedleworkHijacking, dostopano: 3.1.2017.

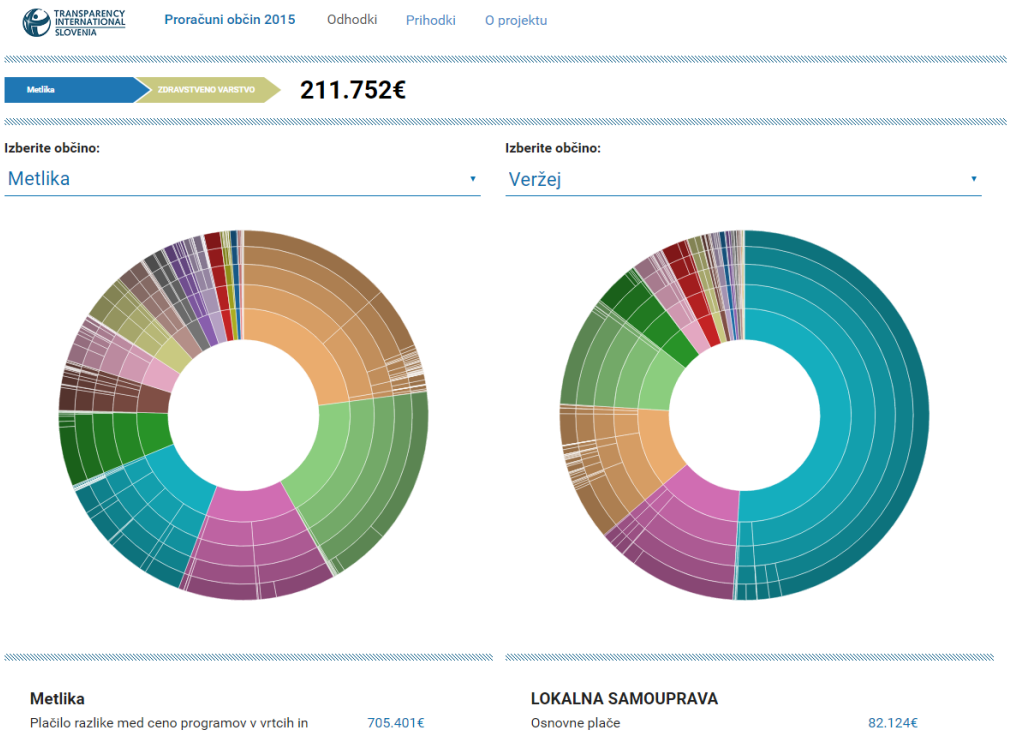
- Aplikacija, namenjena prikazu proračuna Republike Slovenije za leto 2010, na Sliki 1.5 je razviden primer prikaza prihodkov. Ob kliku aplikacija ponudi podroben pogled v vsako kategorijo. Prikaz je statičen ter ne omogoča izbire kriterijev za primerjavo.



Slika 1.5: Aplikacija, namenjena prikazu proračuna Republike Slovenije za leto 2010. Na sliki je viden pogled pregleda prihodkov. Dostopno na: https://static.slo-tech.com/stuff/proracun2010full/index.html, dostopano: 3.1.2017.

- Ena izmed bolj izpopolnjenih aplikacij je aplikacija Transparecny International Slovenija, ki prikazuje proračune občin za leto 2015. Aplikacija z votlimi tortnimi diagrami prikazuje razmerja med posameznimi kategorijami, kamor je bil alociran denar v proračunu občin. Aplikacija ponuja primerjavo dveh občin tako z grafičnega kot tudi s tekstovnega vidika. Ob postavitvi miškinega kazalca uporabnik pridobi dodatne informacije o tipu odhodka oziroma prihodka ter višini le-tega. Ob kliku na posamezen predel aplikacija ponuja podrobnejši ogled področja. Omejitve aplikacije so na primer primerjava le dveh občin,

prikaz na vedno enakem tipu grafa ter prikaz podatka neodvisno od velikosti občine. Primer pogleda aplikacije je na Sliki 1.6.



Slika 1.6: Transparecny International Slovenija. Prikaz odhodkov občin Metlika in Veržej za leto 2015. Dostopno na: http://www.transparency.si/projekti/proracuni-obcin/, dostopano: 3.1.2017.

Poglavje 2

Tehnologije

2.1 Uporabniški del aplikacije

2.1.1 Programski in označevalni jeziki

Pri izdelavi spletne aplikacije na strani uporabnika smo uporabili skriptne ter označevalne jezike. Uporabli smo tehnologije TypeScript, HTML in CSS. TypeScript je skriptni jezik, medtem ko sta HTML ter CSS označevalna jezika.

JavaScript

JavaScript je skriptni jezik, ki ga je izdelal Brendan Eich v desetih dneh. Jezik je bil prvotno imenovan Mocha, kasneje LiveScript ter v končni obliki JavaScript. Sintaktično je JavaScript podoben Microsoftovemu Visual Basic jeziku, Sunovem Tcl oziroma IBM-ovem REXX jeziku. JavaScript je prvotno skriptni jezik, ki ga prevaja uporabnikov brskalnik in kot tak primarno uporabljen za razvoj spletnih aplikacij. V razvoju diplomskega dela smo za izris grafov večinoma uporabljali JavaScript.

Kot omenjeno, je JavaScript skriptni jezik, ki je bil zasnovan za bogatenje vsebine spletnih strani. Njegova primarna naloga je bila generiranje vsebine prilagojene posameznemu brskalniku, vstavljanje dinamičnega HTML ter va-

lidacija podatkov vnešenih s strani uporabnika. Leta 2011 je bil definiran standard ECMAScript 5.1, ki ga podpirajo vsi brskalniki. Prihod standarda je za JavaScript pomenil razvoj v celosten programski jezik, ki ga uporabljamo za razvoj različnih algoritmov in programov. Po nekaterih meritvah (StackOverflow ter GitHub) je JavaScript celo najbolj uporabljen programski jezik.
Njegovi popularnosti pripomorejo prednosti pred ostalimi jeziki, kot so:

- Je edini jezik, ki uživa standardno podporo vseh brskalnikov.
- Je edini univerzalni jezik, v katerem lahko razvijalec razvije celotno aplikacijo z uporabo le JavaScripta. Ostali jeziki so primerni le za zaledne sisteme in potrebujejo podporo JavaScripta na osprednjem delu.
- Kombinira tako objektno usmerjeno kot tudi funkcijsko usmerjeno programiranje.

JavaScript je objektno usmerjen programski jezik, vendar se od ostalih programskih jezikov razlikuje v tem, da nima točno določenega koncepta razreda.

Kot omenjeno, vsi brskalniki podpirajo ECMAScript 5.1 (bolj znan pod okrajšavo ES5) standard. V naši rešitvi smo uporabili novejšega ES6, ki dodaja ogromno novih funkcij, kot so konstante, puščične funkcije, razširjeni funkcijski parametri, definicije razredov in iteratorji v zankah[14].

TypeScript

TypeScript je skriptni jezik, ki izvira iz JavaScript skriptnega jezika. Razvili so ga pri Microsoftu ter predstavlja nadgradnjo JavaScript skriptnega jezika. Sintaksi sta si med seboj skoraj identični. TypeScript omogoča uporabniku uporabo objektno orientiranega programiranja, ki ga poznamo v jezikih kot so C, C++ ali Java. S tem odpravlja šibkosti, ki jih ima JavaScript pri razvoju večjih aplikacij. Omogoča definicijo datotek imenovanih Typings, ki vsebujejo opise JavaScript knjižnic, kar zagotavlja ostalim programom uporabo le-teh, kot da bi bile razvite v TypeScript jeziku[13].

TypeScript se v našem primeru pred izvedbo prevede v JavaScript skriptni jezik standarda ES6 ter zato dovoljuje uporabo JavaScript sintakse.
TypeScript je neodvisen od operacijskega sistema.

HTML

HTML je jezik, ki opisuje strukturo predlog internetnih strani. Sestavljen je iz elementov, ki so gradniki vseh HTML spletnih strani. Elementi so sestavljeni iz značk, s pomočjo katerih definiramo lastnosti elementa[15].

V diplomski nalogi uporabljamo HTML5. Glede na prejšnji standard prinaša veliko novosti, za naše diplomsko delo pa je pomembna podpora umerljive vektorske grafike (angl. scalable vector graphic, v nadaljevanju SVG).

D3.js

Poleg še nekaj manjših knjižnic smo za izdelavo diplomskega dela uporabili tudi knjižnico D3.js, ki skrbi za izris grafov. Kot skupek JavaScript, HTML, SVG in CSS tehnologij nam knjižnica D3.js omogoča pretvorbo podatkov v vizualne predstavitve. Knjižnica vsebuje veliko metod za kreacijo ter manipulacijo HTML elementov, ki jih potrebujemo za izris grafov v naši spletni aplikaciji. Knjižnica D3.js ponuja tudi široko paleto uporabnih funkcij, ki bi razvijalcu vzele velik časovni zalogaj v primeru, da bi jih razvijal sam.

Pri izdelavi diplomskega dela so največjo vrednost predstavljale funkcije, ki skrbijo za poenostavljeno določanje ciljnega HTML elementa, spremembe atributov elementov ter dodajanje nadaljnjih elementov. Uporabili smo tudi naprednejše funkcije, ki na primer skrbijo za izračun pretvorbe iz dejanske vrednosti podatka v dolžino osi pri izrisu grafa[4].

2.1.2 Angular2

Angular2 je ogrodje za izdelavo spletnih aplikacij. Pri izgradnji spletnih aplikacij v tem ogrodju je razvijalcu omogočena uporaba skriptnega jezika

TypeScript. Angular2 aplikacije sestojijo iz komponent. Vsaka komponenta je kombinacija HTML predlog ter komponentnih razredov, ki nadzirajo obnašanje dela zaslona, na katerem se komponenta nahaja. Komponente običajno uporabljajo tako imenovane servise oziroma krmilnike, ki vsebujejo logiko za komuniciranje z zalednim sistemom. V vsaki komponenti lahko uporabimo knjižnice, ki so na voljo na spletu, ter si tako olajšamo delo[1].

CSS

CSS je namenjen slogovnemu oblikovanju spletnih aplikacij in v nekaterih primerih določanju njihovega obnašanja. Brskalnik lahko ob nalaganju HTML datotek slogovne stile najde v samostojnih datotekah s končnicami .css ali pa kot dele samih HTML elementov. Z določanjem stila lahko vplivamo na veliko različnih stvari, kot so velikosti in barve posameznih elementov, pa tudi spremembe ob določenih dogodkih[3].

Bootstrap

Bootstrap je knjižnica pravil, ki nam olajšajo oblikovanje spletne strani. Knjižnica ima dva dela. Skriptni del nadzira programsko logiko, ki skrbi za obnašanje elementov. Drugi del pa je skupek oblikovnih pravil, ki jih uporabnik lahko koristi pri izdelavi svoje HTML predloge. Knjižnica je bila razvita pri podjetju Twitter, kjer so opazili, da njihovi uslužbenci pogosto rešujejo probleme, ki so že bili rešeni[2].

2.2 Strežniški del aplikacije

2.2.1 Programski jeziki

Aplikacija na strežniški strani je v celoti izdelana v skriptnem jeziku TypeScript, ki ga pred izvajanjem prevedemo v skriptni jezik JavaScript.

2.2.2 Node.js

Node.js[10] je JavaScript okolje, ki ga poganja Chromeovo V8 JavaScript jedro. Okolje Node.js pretvori JavaScript skriptni jezik, ki je primarno jezik za izdelavo aplikacij na uporabniški strani, v orodje za izdelavo jedra strežniške strani aplikacije. Node.js uporablja sistem, ki temelji na dogodkih, ter ima vhodno-izhodni sistem, ki ne zaklepa delovanja aplikacije ob morebitnem čakanju. Takšna kombinacija ga naredi hitrega in nezahtevnega za strojno opremo.

Node.js paket vsebuje tudi upravljavca knjižnic npm.

npm

Npm[11] je upravljavec knjžnic za aplikacije, ki temeljijo na Node.js jedru. Npm uporabniku omogoča uporabo že obstoječih knjižnic. Ob prvem zagonu npm kreira datoteko, v kateri so kasneje shranjeni zapisi uporabljenih knjižnic. S to datoteko si razvijalec olajša distribucijo končne aplikacije, saj se odpravi potreba po prenosu uporabljenih knjižnic. Te s posebnim ukazom pridobimo iz spletnega repozitorija. Npm se ponaša z velikim številom uporabnikov, ki posledično pomeni tudi dobro podporo v prihodnosti.

Npm lahko uporabimo v različne namene. Npm ni le upravljavec knjižnic, lahko ga uporabimo tudi za izvajanje določenih ukazov, ki jih definiramo v datoteki, kjer hranimo zapise uporabljenih knjižnic. Npm v našem primeru uporabljamo tudi za zagon serviranja spletne strani.

Typings

Typings so majhne knjižnice, ki omogočajo lažjo uporabo JavaScript knjižnic, ko uporabljamo TypeScript. Vsebujejo le opise knjižnic JavaScript in ne dodajajo novih funkcionalnosti.

2.2.3 MongoDB

Pri izdelavi aplikacije smo uporabili nerelacijsko bazo, ki temelji na skupku dokumentov. Dokumenti so v JSON obliki in nimajo povezav z ostalimi dokumenti. Po podatkovni bazi iščemo s parom ključ in vrednost. Takšen način iskanja ponuja hitrejše iskanje po bazi, kot rezultat pa dobimo vse dokumente, ki vsebujejo podane kriterije. Za posamezen dokument baza ne vsebuje nikakršne definicije, torej bi lahko imeli v končni bazi povsem različne dokumente. Ker dokumenti niso definirani, lahko pri razvoju aplikacije določimo njeno obnašanje že na podlagi rezultatov, ki jih pridobivamo skozi zaledni sistem[8, 9].

2.3 Razvojno okolje

Za razvoj diplomskega dela smo si pomagali z razvojnim okoljem JetBrains WebStorm 2016.3.1.

Za razvijanje spletne aplikacije in njenega zalednega sistema razvijalec potrebuje določena orodja. Uporaba posameznih orodij za posamezne naloge lahko pripelje do nevšečnosti zaradi števila posameznih aplikacij. WebStorm je okolje, ki večino orodij združuje v enem razvojnem okolju.

Glavna orodja, ki smo jih uporabljali pri izdelavi diplomskega dela so:

- **Urejevalnik besedila**, ki je osrednje okno, namenjeno pisanju kode programa. Lahko se deli na več oken po vertikali ali horizontali, kar pripomore k lažjemu razvijanju aplikacije.

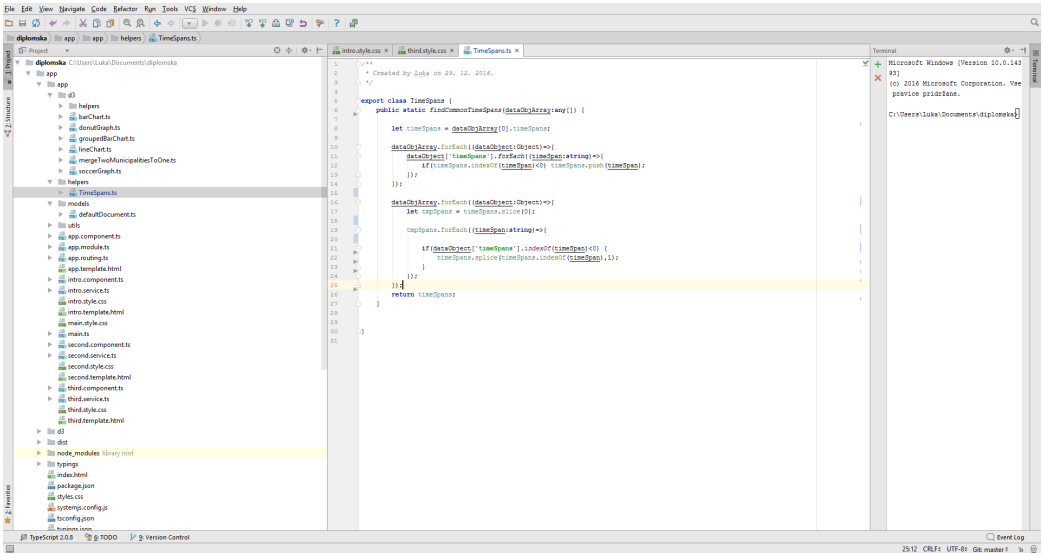
- **Projektno okno (angl. Project)**, v katerem se nahaja hierarhija projekta. Okno je uporabno za hitro ustvarjanje novih datotek in za upravljanje s kontrolo verzij.

- **Terminal** je integriran v okolje ter nadomesti terminal, ki bi ga sicer priskrbel operacijski sistem. Integracija terminala v razvojno okolje pripomore k hitrejšemu razvoju, saj lahko hitreje dodajamo knjižnice,

poganjamo aplikacijo ali pa spremljamo njen zagon ob morebitnih spremembah kode. Razvijalec tako nemudoma vidi, če je ob spreminjanju kode prišlo do napake v prevajanju ali zagonu aplikacije.

- **Zgornja orodna vrstica**, ki nudi dostop do vseh orodij, ki jih dana aplikacija ponuja.
- **Spodnja orodna vrstica**
  - **Prevajalnik (angl. Compiler)** je prvo izmed pojavnih oken, v spodnji orodni vrstici. V našem razvojnem okolju je to prevajalnik skriptnega jezika TypeScript. Prevajalnik nam ob možnostih prevajanja posamezne datoteke ali celotnega projekta nudi še izpis konzole ter izpis napak, do katerih je morebiti prišlo pri prevajanju kode.
  - **Kontrola verzije (angl. Version control)** nam omogoča hitro uporabo kontrole verzij s poljubnim orodjem. Za razvoj diplomskega dela smo uporabili GIT kontrolo verzij, temu primerna je samodejna nastavitev okna kontrole verzij.

Prikaz uporabljenega razvijalnega okolja je na Sliki 2.1:



Slika 2.1: Primer razvojnega okolja Webstorm. Leva stran predstavlja projektno okno, sredinski del zavzema urejevalnik besedila, desni del pa okno vgrajenega terminala. Spodnji del okolja zaseda vrstica z bližnjicami do prevajalnika ter kontrole verzij.

Poglavje 3

Izdelava aplikacije

S tehnologijami opisanimi v prejšnjem poglavju smo želeli doseči zastavljen cilj diplomskega dela: predstavitev odprtih demografskih podatkov z vizualizacijo, ki bo uporabniku razumljiva in preprosta.

V ta namen smo razvili aplikacijo ki vsebuje:

- Zaledni del z metodami za:
  - iskanje podatkov
  - uvoz podatkov
  - iskanje po podatkih
  - prvotno obdelavo podatkov
  - modeli spremenljivk.
- Uporabniški del, ki ga predstavlja spletna aplikacija, in obsega:
  - Prvi pogled s pripadajočimi krmilniki, HTML predlogami in stili, ki predstavljajo uvodno stran v aplikacijo.
  - Drugi pogled s pripadajočimi krmilniki, HTML predlogami in stili, ki uporabniku ponudijo poenostavljeno različico vmesnika. Pogled je namenjen predvsem uporabnikovem spoznavanju z delovanjem aplikacije.

– Tretji pogled s pripadajočimi krmilniki, HTML predlogami in stili, ki tvorijo glavni del aplikacije. Pogled uporabniku ponuja popoln nadzor nad upravljanjem vizualizacije.

– Izris grafov, ki je del aplikacije uporabljen v dveh pogledih, vendar ga je zaradi njegove pomembnosti smotrno izpostaviti.

3.1 Izdelava strežniškega dela aplikacije

3.1.1 Iskanje podatkov

Izdelavo diplomskega dela smo začeli s pridobivanjem podatkov, ki so bili kasneje dodani v našo podatkovno bazo. Podatke smo v celoti pridobili s spletne strani Statističnega urada Republike Slovenije. Podatki, ki so bili v celoti v odprti obliki, so bili izvor ideje za izdelavo spletne aplikacije. Pri prenosu podatkov smo ugotovili, da podatki niso zagotovljeni za ista časovna obdobja. Tako so podatki za število prebivalcev posamezne občine zabeleženi za dve leti, podatki za število prebivalcev brez izobrazbe pa za obdobje petih let. Prav tako je bila oblika, ki jo ponuja statistični urad ob izvozu, neprimerna za uporabo v naši aplikaciji.

3.1.2 Uvoz podatkov

Za uvoz podatkov v podatkovno bazo smo razvili skripto, kjer podamo lokacijo datoteke, ki jo želimo uvoziti, nato pa jo poženemo s pomočjo Node.js okolja. Skripta je pomemben dodatek k rešitvi, saj omogoča njeno razširitev na katerikoli tip podatkov. Podatke moramo prirediti strukturi za uvoz v našo aplikacijo. Prva vrstica datoteke mora vsebovati tip oziroma naslov podatka, ki ga uvažamo. Druga vrstica mora vsebovati časovne odseke, ki jih podatki predstavljajo. Časovni odseki, natančneje letnice, morajo biti tako kot podatki v nadaljevanju datoteke ločeni s podpičjem. Vse naslednje vrstice vsebujejo podatke za posamezne občine. Vsaka vrstica predstavlja svojo občino. Začetek vrstice je lahko prazen prostor označen z nareko-

vaji in z narekovaj označeno ime občine. Vrstica se nadaljuje s podatki, ki pripadajo posameznim časovnim odsekom navedenim v prvi vrstici. Tako strukturo podatkov smo določili glede na strukturo izvoza podatkov, ki jih ponuja spletna stran Zavoda za statistiko Republike Slovenije. Skripta ob zagonu pregleda, če navedena datoteka obstaja ter jo nato prebira vrstico za vrstico. Po vsaki končani vrstici skripta ustvari dokument, ki predstavlja eno občino. Sestavljanje poteka tako, da skripta vzame prvi dve vrstici, ki vsebujeta tip podatka in časovne odseke ter trenutno vrstico, ki vsebuje ime občine in dejanske podatke za podan tip. Skripta pretvori vrstice v dokument oblike JSON in ga vstavi v podatkovno bazo. Po pregledu datoteke, če je le-ta bila v pravilnem formatu, je skripta odstranila vse posebne znake, ki bi jih vrstice lahko vsebovale, ustvarila ter vstavila dokumente za podani tip podatkov.

3.1.3 Iskanje po podatkovni bazi

Zaledni sistem je zadolžen tudi za iskanje po podatkovni bazi. Ob prejetemu zahtevku, poslanem preko HTTP, sproži metodo, ki ustreza prejetemu zahtevku. Jedro zalednega sistema opravi vse preusmeritve glede na tip zahtevka. Metoda, ki se bo izvršila, je odvisna od spletnega naslova kamor je bil zahtevek poslan. V našem primeru je bilo razvojno okolje lokalno, zato je bil naslov strežnika *http://localhost:3000*. Pot, ki sledi naslovu strežnika določa nadaljnji potek. Tako je zahtevek poslan z GET metodo na naslov *http://localhost:3000/municipalityNames* vrnil rezultat, ki je vseboval vsa imena občin v podatkovni bazi. Jedro vsebuje več različnih metod, ki vračajo podatke glede na število občin, število kriterijev, normalizacijo po kriteriju ter kombinacije prej naštetih. Vse metode najprej vzpostavijo povezavo s podatkovno bazo, ter nato pošljejo poizvedbo, ki jo kreira metoda. Za primer omenimo dve metodi:

- **SingleDoc.getSingleDoc()** je metoda, ki se izvrši ob poslani GET poizvedbi na naslov *http://localhost:3000/singledoc /:name*. Zadnji del poizvedbe (:name) zamenjamo z imenom občine,

po kateri poizvedujemo. Jedro bo iz parametrov poizvedbe pridobilo atribut *name*, ki predstavlja ime občine, ter ga posredovalo metodi. Metoda bo vzpostavila povezavo s podatkovno bazo ter ji poslala poizvedbo po dokumentih, ki vsebujejo podan atribut. Rezultat metode je JSON dokument, ki predstavlja eno občino s podatki za vse podatkovne tipe.

- **SingleDoc.getSingleDocNormalized()** je primer druge metode, ki ob zahtevku po dokumentu zagotovi še normalizacijo glede na določen podatkovni tip.

3.1.4 Obdelava podatkov

Metoda *SingleDoc.getSingleDocNormalized()* lahko služi tudi kot primer metode, ki obdeluje podatke na strežniškem delu aplikacije. Za primerjavo podatkov na uporabniški strani je uporabniku smiselno omogočiti primerjavo med normaliziranimi podatki. Odločili smo se, da bo normalizacija podatkov potekala na zalednem delu aplikacije, saj tako ne obremenjuje uporabnikovega brskalnika. Vsakokratno normaliziranje podatkov prihrani tudi prostor, saj ne shranjujemo normaliziranih podatkov za vsak podatkovni tip. Metode za vračanje normaliziranih podatkov tako večinoma vsebujejo klic metod, ki vračajo dokument za določen tip podatkov, ter logike, ki z danimi podatki sestavi pričakovani rezultat. Na začetku podpoglavja omenjena metoda kot atribute dobi ime občine, za katero želimo normalizirane podatke, podatkovni, tip po katerem želimo podatke normalizirati ter tabelo podatkovnih tipov, za katere želimo podatke. Metoda nato za vsak element v tabeli pošlje poizvedbo za normaliziran dokument. Vsak vrnjen dokument metoda vstavi v tabelo, ki jo na koncu vrne kot rezultat.

3.1.5 Modeli spremenljivk

Z uporabo TypeScript jezika pridobimo možnost, da vsaki spremenljivki točno dolčimo njen tip, česar sicer JavaScript ne omogoča. TypeScript ima

določene štiri: glavne tipe niz (angl. String), število (angl. Number), objekt (angl. Object) ter katerikoli (angl. any). Poleg tega pa nam omogoča definicijo novih tipov. Tipe lahko definiramo kot vmesnike ali pa kot razrede, ki so pogosteje uporabljeni pri razvijanju z jezikom Java. Za uporabo na strežniški strani aplikacije smo definirali model dokumenta, ki ga vračamo kot rezultat metod. Ker se model uporablja kot tip objekta, smo ga razvili kot razred, ter temu primerno definirali konstruktor ter atribute. Konstruktor je prazen, saj ne želimo ob kreaciji spreminjati ničesar, atributi pa sovpadajo z definicijo v podatkovni bazi. Zajemajo ime občine, tip podatka, ki ga dokument hrani, tabelo časovnih enot, za katere hranimo podatke in dejanske podatke. Kot zanimivost omenimo, da v TypeScript tehnologiji ne potrebujemo metod za nastavljanje (angl. setter) in pridobivanje (angl. getter) podatkov.

3.2 Izdelava uporabniškega dela aplikacije

Za izdelavo uporabniškega vmesnika smo uporabili ogrodje Angular2. Z uporabo tega ogrodja razvijamo spletno aplikacijo kot aplikacijo na eni strani, znotraj katere menjavamo vsebino. Vsebino opisujemo v komponentah, ki jih lahko uporabimo večkrat. Komponentam dodamo HTML predlogo, datoteko s stili ter krmilnik komponente. Aplikacija se tako deli na sledeče dele:

- app.component komponento,
- intro.component komponento,
- second.component komponento,
- third.component komponento,
- izris grafov s knjižnico D3.js.

3.2.1 Oblikovanje uporabniškega vmesnika

Uporabniški vmesnik je ključni del spletne aplikacije. Vmesnik uporabniku ponuja način komunikacije z računalnikom. Tako mu omogoča izkoriščanje

logike spletne aplikacije. Pri izdelavi grafičnega vmesnika smo sledili trem principom oblikovanja vmesnikov. To so principi razumnosti, uporabnosti in konsistentnosti.

Princip razumnosti temelji na korektni postavitvi gradnikov vmesnika glede na logiko uporabnikovega miselnega procesa. Kreativnost je pri oblikovanju dobrodošla, vendar pa moramo paziti, da ne odstopamo preveč od ustaljene oblike vmesnikov. Prevelika odstopanja od ustaljenih praks lahko povzročijo zmedenost pri uporabniku.

Princip uporabnosti predstavlja uporabnost kot temelj oblikovanja vsake aplikacije. Aplikacija mora biti efektivna, učinkovita, tolerantna za napake uporabnika, lahka za razumevanje ter privlačna uporabnikovem očesu.

Pri razvoju uporabniškega vmesnika moramo paziti na konsistentnost. Uporabniški vmesnik mora biti konsistenten pri zagotavljanju intuitivnosti ter prijaznosti uporabe skozi celotno uporabniško izkušnjo[5].

3.2.2 App komponenta

App komponenta je poseben del aplikacije, saj za razliko od ostalih komponent nima tako imenovanega krmilnika. Glavna naloga komponente je zagotavljanje modulov za celotno aplikacijo. Prav tako je edina izmed komponent, ki ima datoteko za preusmerjanje (angl. routing), ki določa poti do ostalih komponent, na katere bodo ustvarjene povezave v spletni aplikaciji. HTML predloga za to komponento predstavlja začetno ogrodje elementov, v našem primeru pa samo poseben element *<router-outlet>*, ki določa, kje bodo prikazane ostale komponente. Če bi obstajale še stilska datoteka, bi določala stil, ki ga dedujejo vse ostale komponente.

3.2.3 Prva komponenta

Vstopna komponenta, poimenovali smo jo intro.component, je komponenta, ki jo uporabnik vidi najprej. Komponenta je namenjena uvodu v aplikacijo. Izmenično prikazuje podatek naključnega podatkovnega tipa za naključno

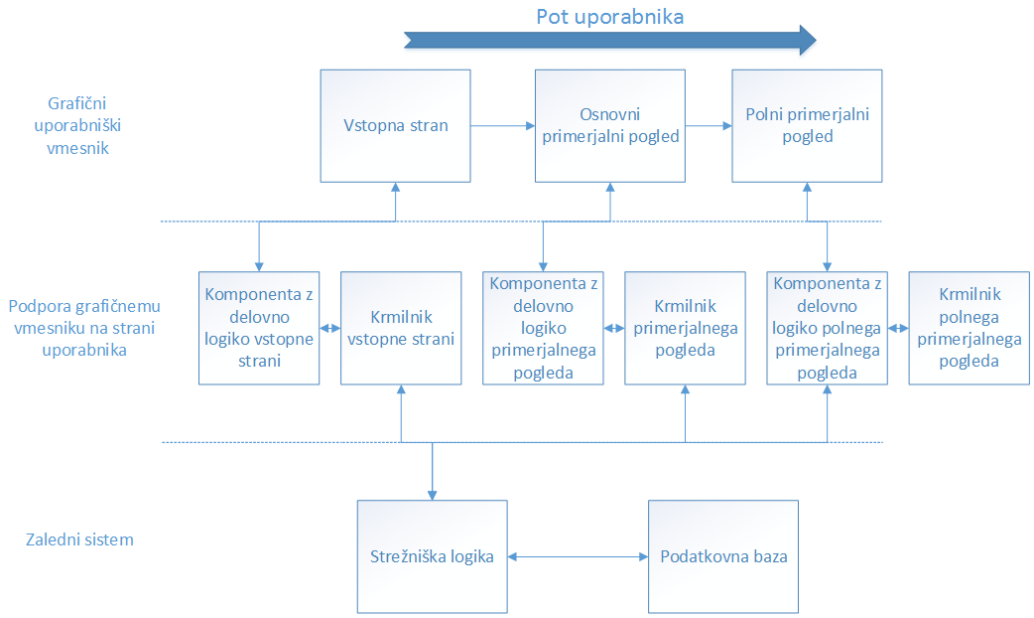
občino. Začetna komponenta je sestavljena po običajnem modelu, kjer komponenti z logiko dodamo HTML predlogo, CSS datoteko s stilom ter krmilnik za komuniciranje z zalednim sistemom.

Komponenta z delovno logiko

Kot vstopna točka v aplikacijo ima komponenta preprosto delovanje. Ob zagonu komponente ta vzpostavi povezavo s krmilnikom ter od njega zahteva prvi dokument za prikaz. Iz dokumenta pridobi podatke ter jih umesti v besedilo, ki se nato izpiše na uporabnikovem zaslonu. Preostanek logike je metoda, ki na vsakih šest sekund zamenja podatke.

Krmilnik

Krmilnik oziroma servis (angl. service) skrbi za povezavo z zalednim sistemom. Krmilnik vstopne točke je zelo preprost. Vsebuje eno metodo, ki od zalednega sistema zahteva podatke prirejene za izpis na vstopni točki. Od zalednega sistema pridobi obljubo (angl. promise). Obljuba je objekt za predstavitev vrednosti, ki je že ali pa še bo na voljo, na katero se prijavi komponenta. Preden krmilnik vrne podatke jih pretvori v obliko JSON. Celotna struktura komponent aplikacije je vidna na Sliki 3.1.



Slika 3.1: Prikaz strukture komponent. Slika predstavlja strukturo dejanske aplikacije in se deli na tri nivoje: grafični uporabniški vmesnik, podpora grafičnemu vmesniku, ki se izvaja na strani uporabnika ter zaledni sistem. Povezave med objekti na prvem nivoju predstavljajo dejanske povezave med pogledi in se vršijo po predvideni poti uporabnika. Pot uporabnika ponazarja puščica nad diagramom. Povezave, ki povezujejo objekte na različnih nivojih ter objekte drugega in tretjega nivoja, predstavljajo pretok podatkov.

3.2.4 Druga komponenta

Druga komponenta, imenovana second.component, skrbi za spoznavanje uporabnika z vmesnikom. Ponuja dve opciji za izbiro občin ter eno opcijo za izbiro podatkovnega tipa za izpis.

Komponenta z delovno logiko

Delovna logika druge komponente ima obrise polne delovne logike, saj je za pravilno delovanje na uporabniški strani potrebno več metod. Komponenta ob zagonu vzpostavi povezavo s krmilnikom, od katerega pridobi dva dokumenta. Ob zagonu aplikacija potrebuje tabelo vseh imen občin v podatkovni bazi ter tabelo vseh podatkovnih tipov, za katere hranimo podatke.

Metodi sta izvedeni ob zagonu (angl. on initialization), podatki, ki jih z njima aplikacija pridobi, pa tvorijo spustne menije za nadzor komponente. Ob ostalih manjših metodah se v drugi komponenti prvič pojavi tudi metoda za risanje grafov (*redraw()*). Izris grafov se izvrši ob vsaki uporabniški spremembi na aplikaciji, kar uporabniku zagotavlja občutek odzivnosti.

Metoda *redraw()* je v drugi komponenti največja metoda, saj skrbi za dotok podatkov in klice pravilnih metod za izris grafov. Metoda pregleda spremenljivke, ki predstavljajo izbire uporabnika, ter glede na njihove vrednosti določa zahteve za podatke, ki se vizualizirajo. Iste spremenljivke so v tej metodi uporabljene v logiki kode, ki prikazuje ali skriva gumbe, s katerimi uporabnik aplikaciji sporoči ali želi vizualizacijo na enem ali na več grafih.

Krmilnik

Kompleksnost krmilnika narašča s kompleksnostjo komponente. Poleg osnovnih klicev za poizvedbe o imenih občin in tipih podatkov, je krmilnik druge komponente odgovoren tudi za zahtevke o poizvedovanju za določene dokumente s podatki in klice za izrisov grafov.

3.2.5 Tretja komponenta

Tretja komponenta, poimenovana third.component, je jedro naše rešitve. Uporabniku omogoča poljubno določanje števila občin ter podatkovnih tipov za primerjavo. Omogoča različne načine primerjav ter dvonivojski prikaz grafov. Številčnost možnosti se odraža v zapletenosti sestavnih delov, ki tvorijo glavni del aplikacije.

Komponenta z delovno logiko

Glavna komponenta ob zagonu zahteva imena občin ter podatkovne tipe, ki so na voljo v podatkovni bazi. Ker je uporabniku na voljo le en spustni meni, s katerega izbira občine ter jih dodaja na listo primerjanih, ima ta komponenta dodane metode za upravljanje z listo izbranih občin. Različne metode, ki nadzorujejo izbiro občin skrbijo, da se izbire ne podvajajo ter da so podatki na voljo metodam, ki bi jih lahko potrebovale. Komponenta ima podobno logiko tudi za delo z izbiro podatkovnih tipov, po katerih želi uporabnik primerjati občine. Večino komponente sestavlja metoda *redraw()*, ki skrbi za izris vizualizacije ob uporabnikovih spremembah.

Metoda za izris je obsežna, saj zajema vse različne scenarije, ki se lahko tvorijo ob različnih kombinacijah uporabnikovih izbir. Metoda nadzira dotok podatkov glede na število izbranih podatkovnih tipov v kombinaciji s številom izbranih občin. Prva odločitev metode temelji na velikosti intervala, za katerega je krmilnik posredoval podatke. V primeru, da je interval omejen na eno enoto (najpogosteje se to zgodi v primeru normalizacije), se uporabniku kot izris vizualizacije prikaže gručni stolpčni graf. Tak graf ima namesto časovne enote na vodoravni osi podane skupine podatkovnih tipov za vrednosti katerih so izrisani stolpci. V primeru, kjer je interval večji od ene enote se uporabniku ponudi izris v obliki linijskega grafa. Metoda izrisa takšnega grafa kot pomožen atribut prejme objekt, ki predstavlja celotno tretjo komponento. Posredovanje takšnega objekta zagotavlja, da ima aplikacija posnetek stanja pred prehodom v drugi nivo grafa. Pomnjenje stanja je pomembno, saj moramo zagotoviti možnost vrnitve na višji nivo prikaza

vizualizacije.

Krmilnik

Krmilnik tretje komponente omogoča uporabo vseh metod, ki jih ponuja zaledni sistem. Tako komponenti zagotavlja vse potrebne podatke. Poleg vseh klicev za pridobitev različnih podatkov krmilnik zagotavlja tudi klice za izris vseh vrst grafov.

3.3 Izris grafov s knjižnico D3.js

3.3.1 Teorija grafičnega prikazovanja

Za dosego cilja diplomskega dela, ki poskuša uporabniku prikazati podatke na najbolj jasen ter nedvoumen način, smo ob implementaciji metod za izris grafov morali slediti določenim načelom ter dobrim praksam vizualiziranja.

Cilji vizualizacije

Vizualizacija ali upodobitev pomeni grafični prikaz informacij. Vizualizacija lahko informacije podaja za dosego različnih ciljev.

Prvi izmed teh je raziskovanje in analiza podatkov za podporo odločitvam. Končnemu uporabniku torej podajamo vizualno predstavitev podatkov, ki bo služila kot pomoč pri nekem procesu raziskovanja podatkov, iskanja vzorcev in napak. Vizualizacijo lahko uporabimo tudi ko želimo uporabniku samo predati informacije, ki z vizualno razlago postanejo lažje ter hitreje razumljive.

Principi oblikovanja vizualizacij

Vizualizacija podatkov lahko ob neprimerni uporabi zavede končnega uporabnika, zato smo ob razvijanju naših implementacij sledili dobrim principom, ki jih je definiral E. Tufte[12]:

- **Razmerje podatki-črnilo** določa koeficient uporabljenega črnila za podatke ter celotne količine uporabljenega črnila. Ob vizualiziranju podatkov želimo doseči da je koeficient čim večji.

- **Izogibanje nepotrebnim elementom** je pomembno, da ob izrisu vizualizacije ne pretiravamo z dodanimi elementi, kot so pomožne črte po vertikali ali horizontali, saj zakrivajo pravo informacijo grafa.

- **Povečanje gostote podatkov** pomeni, da želimo podatke strniti na čim manjšo površino, pri čemer je potrebno paziti, da so podatki še vedno lahko berljivi. Kot pri razmerju podatki-črnilo tudi tu lahko izračunamo koeficient med številom prikazanih podatkov ter površino podatkov v grafiki.

- **Ločevanje na nivoje** uporabniku omogoča dodatni vpogled v same podatke in tako lažjo pridobitev informacije.

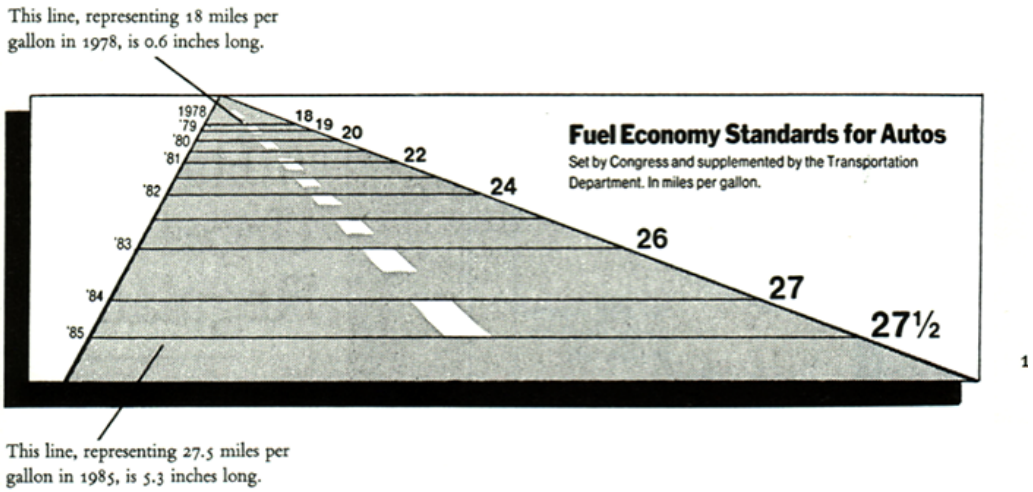
Slaba vizualizacija

Ob izdelavi vizualizacije je potrebno tudi znanje kdaj, je vizualizacija slaba. Napačna vizualizacija lahko spremeni uporabnikovo zaznavanje razmerja podatkov ter tako sugerira informacijo, ki je podatki ne podajajo. Takšne vizualizacije so pogosto uporabljene kot način podpiranja napačnih informacij, namenjenih, da zavedejo oziroma prepričajo uporabnika v resničnost le-teh. Takšne vizualizacije pogosto uporabljajo prirejene skale. Percepcijo razlik med podatki povečamo ali pomanjšamo z uporabo skal, ki se ne začenjajo z nič. Tako lahko prikažemo podatke o povečanju nekega števila s skalo, ki se začne z vrednostjo različno od nič, konča pa s povečanjem te vrednosti za neko minimalno razliko. Velikost vizualne razlike v grafu tako podaja informacijo o znatnem povečanju, medtem ko lahko dejanski podatki kažejo na minimalno povečanje.

Informacije lahko priredimo tudi s popačenjem glede na skalo, tako da skala na eni strani prikazuje določene mere na drugi strani pa te mere ra-

hlo povečamo. S povečanjem skale na drugi strani tako dosežemo napačno interpretacijo informacij.

Na pridobivanje napačnih informacij lahko vplivamo tudi ob uporabni nestandardnih vizualizacij, kot so prikazi razmerja podatkov glede na spremembo površine. Ilustracije, ki prikazujejo spremembe v podatkih lahko uporabnika zavajajo, če je razlika med grafičnimi segmenti vizualizacije različna od razlike v podatkih. Primer je viden na Sliki 3.2.



Slika 3.2: Primer zavajanja z ilustracijo spremembe podatkov.

Tako deviacijo lahko merimo s faktorjem laži. Faktor laži se meri kot razmerje med prikazom efekta v ilustraciji ter razmerjem efekta v podatkih, kjer je razmerje enako absolutni vrednosti razlike med drugo in prvo vrednostjo podatka deljeno z vrednostjo prvega podatka. Za ohranitev integritete grafičnega prikaza naj bi bil njen faktor laži v intervalu med 0,95 ter 1,02. Primer na Sliki 3.2 ta faktor krepko presega, kar je razvidno že na prvi pogled.

Vse izpostavljene primere lahko strnemo v Tuftejeve principe poštenosti[12]:

- prikazuj spremembe v podatkih, ne obliki,
- uporabljaj jasne in natančne oznake ter primerne skale,

- velikost grafičnega učinka naj bo enakovredna podatkovnim vrednostim.

3.3.2 Osnove knjižnice D3.js

Za izris grafov je bila uporabljena knjižnica D3.js. Knjižnica je bila razvita, da uporabniku olajša delo s SVG elementi. Umerljiva vektorska grafika (angl. scalable vector graphic) je pogosto uporabljena pri spletnih aplikacijah za izris različnih elementov. Pri izdelavi takih aplikacij lahko z manipulacijo osnovnih oblik razvijalec doseže izris praktično vsake oblike, kar je tudi glavni koncept knjižnice D3.js. Knjižnica nam omogoča, da na preprost način izberemo kateri koli element spletne strani ter nato z njim upravljamo. Na element lahko pripenjamo nove elemente, mu določamo stile ali pa celo dodajamo razrede. Izris grafa je tako le zaporedje dodajanja elementov.

Izbira elementov

Knjižnica vsebuje metodo za hitro in enostavno izbiranje elementov preko celotne spletne strani. Posamezen element izberemo z metodo *d3.select()*. Primer: *d3.select("#id")*. Z enako metodo lahko izberemo tudi vse elemente z določeno značko. Primer izbire vseh odstavkov *d3.select("p")*. Zaradi hitrejše uporabe izbire elementov pogosto shranjujemo v spremenljivke.

Veriženje ukazov

Zelo pomemben koncept za razumevanje knjižnice je veriženje ukazov. Kot smo že omenili, je izris grafa nekakšno zaporedje pripenjanja ter upravljanja z elementi. Vsak ukaz lahko pripnemo na prejšnjega ter tako ustvarimo strukturo kode, ki je sama po sebi jasna in razumljiva. Osnovni primer veriženja je pripenjanje elementa na drugega, ki v kodi izgleda takole:

```
1 d3.select("\#graphContainer")
2 .append("svg");
```

Primer kode 3.1: Primer veriženja ukazov.

Upravljanje z elementi

Za uspešno implementacijo izrisa je najprej potrebno poznavanje upravljanja z elementi. Knjižnica nam omogoča, da kateremu koli elementu pripnemo novega. Tako lahko na risalno površino tipa SVG pripnemo pravokotnik, ki bi lahko predstavljal stolpec grafa. Pripenjanje elementov izvajamo z veriženjem izbire ter pripenjanja.

Elementom lahko poleg pripenjanja drugih elementov pripnemo tudi poslušalce dogodkov (angl. event listener), ki omogočajo interaktivnost spletne strani. Kot primer vzemimo spremembo prosojnosti okna ob premiku miškinega kazalca na element prepoznan kot *container*:

```
1 container.on("mouseover", function (d){
2   d3.select(this).style("opacity", .5);
3 });
```

Primer kode 3.2: Poslušalec ukazov.

Elementu lahko s podobnimi ukazi spreminjamo vse atribute.

Osnovne oblike

Za izris grafov smo uporabili osnovne oblike vektorske grafike. Na risalno podlago smo dodajali pravokotnike, kroge, črte in poti. Obliko dodamo kot vsak element z metodo *.append()*.

Dodajanje oblik s podatki

Verjetno najbolj zapleten koncept knjižnice je dodajanje oblik glede na podatke. Manjšo zmedo povzroča začetna izbira elementov, ki označuje elemente, ki so s podatki dodani po izbiri. Primer dodajanja pravokotnikov s podatki:

```
1  var circles = d3.select("svg").selectAll("rect")
2   .data(data)
3   .enter()
4   .append("rect");
```

Primer kode 3.3: Dodajanje oblik.

V primeru najprej izberemo SVG element, ki ga priredimo spremenljivki *circles*. Nato označimo vse pravokotnike, ki jih šele kasneje ustvarimo. Verižen ukaz *.data(data)* iz spremenljivke data prebere podatke, ukaz *.enter()* pa metodi pove, da glede na te podatke pripenja pravokotnike. Metoda za vsak podatek v spremenljivki pripne en pravokotnik, ki hrani podatke iz tabele, za katere je bil pripet.

Prirejanje podatkov

Zadnji izmed pomembnejših konceptov je skaliranje podatkov (angl. scaling), katerega potrebujemo za pravilen prikaz razmerja podatkov z grafiko. Knjižnica vsebuje različne metode, med katerimi smo uporabili slednje:

- Metoda *d3.scaleLinear()*, ki linearno pretvarja podatke, uporabljena največkrat za vertikalno os, torej za prikaz linearnih vrednosti. Linearno pretvarjanje podatkov lahko uporabimo tudi za določanje barv, kjer kot začetek in konec določimo barvi, med katerimi nam metoda kot rezultat vrača barvni odtenek.

- Metoda *d3.scaleBand()*, ki iz podatkov gradi pasove, je uporabljena pri izdelavi skal za skupine.

- Metoda *d3.scaleTime()*, ki iz podatkov zgradi skalo po času.

Vsaki skali določamo širino in domeno. Širina skale predstavlja dejansko vrednost na površini izrisa, medtem ko domena predstavlja razpon podatkov.

3.3.3 Implementacija izrisa linijskega grafa

Največkrat izrisan ter tako osrednji graf aplikacije je linijski graf (angl. line chart). Linijski graf v aplikaciji izrisujemo v drugi in tretji komponenti. Krmilnika obeh komponent imata v svojo logiko uvožen razred, ki definira linijski graf ter ob zahtevi komponente kličeta metodo *drawLineChart()*.

Metoda za izris linijskega grafa najprej določi odmike grafa od roba risalne površine, na podlagi katerih nato izračuna velikost grafa. Po izračunu velikosti grafa je naslednja naloga metode izračun velikosti skal, ki se prilagajajo risalni površini grafa. Tako določimo dolžino horizontalne osi grafa od izhodiščne točke risalne površine grafa do konca le-te. Za izris časovnih osi uporabimo metodo, ki jo ponuja knjižnica D3.js. Podobno velja za vertikalno os, za katero določimo interval, po katerem bomo linearno razporedili vrednosti podatkov. V tem delu definiramo interval, iz katerega bo kasneje metoda pridobivala podatke o barvah, ki jih bo uporabila za barvanje različnih delov grafa. Za izris linijskega grafa moramo podatke, ki jih je posredovala komponenta, preurediti v obliko, s katero lahko izrisujemo graf s pomočjo knjižnice D3.js.

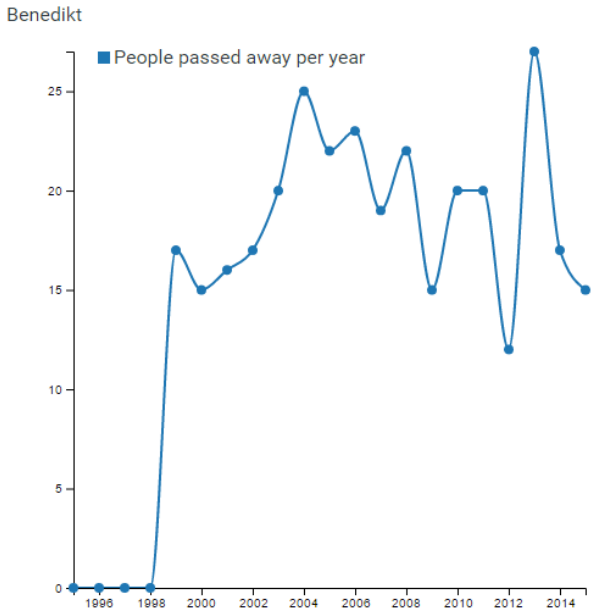
Za preurejanje podatkov smo razvili posebno skripto, ki tabelo dokumentov najprej razstavi na posamezne dokumente po tipu podatkov. Skripta nato spremeni ključe, po katerih pridobivamo vrednosti, tako da jih kasneje lahko uporabimo za izris.

Po pridobitvi podatkov lahko obema osema določimo tudi domeno, v katero bodo razporejeni podatki. Ko imamo postavljeno ogrodje za izris grafa, je potrebno še zadnjič preveriti ali je končne podatke še vedno smiselno predstavljati kot linijski graf. Če je interval podatkov dolg eno enoto, namesto linijskega grafa raje izrišemo gručni stolpčni graf. Zaradi časovne enote omejene na eno enoto (leto) na horizontalno os raje izpišemo skupine grafov, ki jih določajo uporabnikove izbire kriterijev za izpis podatkov. V ostalih primerih metoda nadaljuje z izrisom linijskega grafa. Po preureditvi podatkov je v spremenljivki, v katero smo podatke shranili, tabela, ki kot posamezen element vsebuje podatke potrebne za izris ene črte grafa. Za vsak element

tabele tako na graf pripnemo pot sestavljeno iz točk, izračunanih glede na podatke. Poti nato določimo širino izrisa ter barvo, ki jo pridobimo iz skale definirane na začetku metode. Za vsak par podatkov (vrednost, čas) na pot dodamo piko. Pot izrisana med pikami je namreč le predvidevanje spreminjanja podatkov.

Interaktivnost grafa smo dosegli z implementacijo poslušalcev dogodkov, ki smo jih pripeli na pike. Ob premiku miškinega kazalca na piko, se ta razširi, notranji del se obarva črno, da dodatno izpostavi dogajanje, poleg miškinega kazalca pa se izriše okno z obvestilom. Prikazano obvestilo z nasvetom (angl. tooltip) vsebuje izpis podatkov, ki jih pika predstavlja. Na obvestilu so torej ime občine, za katero preverjamo podatke, leto zabeleženih podatkov ter vrednost le-teh. Če podatkov ni, izrisujemo namreč vedno glede na največji interval po vseh podatkih in to se odraža v pojavljenem sporočilu. Ob kliku na piko uporabnik preide na naslednji nivo vizualizacije, ki je izražena primerno podanim podatkom.

Na koncu izrisa metoda, glede na podatke, ki jih je pridobila od komponente, grafu doda legendo. Legenda je sestavljena iz kvadratov pobarvanih z enako barvo kot pripadajoča pot ter tekstom, ki predstavlja naslov poti. Metoda na vrh grafa doda tudi naslov grafa za lažje razumevanje izrisa. Preko grafa je postavljen pravokotnik v dimenzijah njegove celotne risalne površine, ki se po izrisu grafa postopoma krči proti desni strani. Krči se v desni smeri dokler kvadrat povsem ne izgine in daje uporabniku občutek animacije grafa. Animacija se izvede v obliki postopnega izrisa poti po časovnem zaporedju za katerega so podani podatki. Primer linijskega grafa za prikaz števila umrlih ljudi v letu za občino Benedikt, je viden na Sliki 3.3.

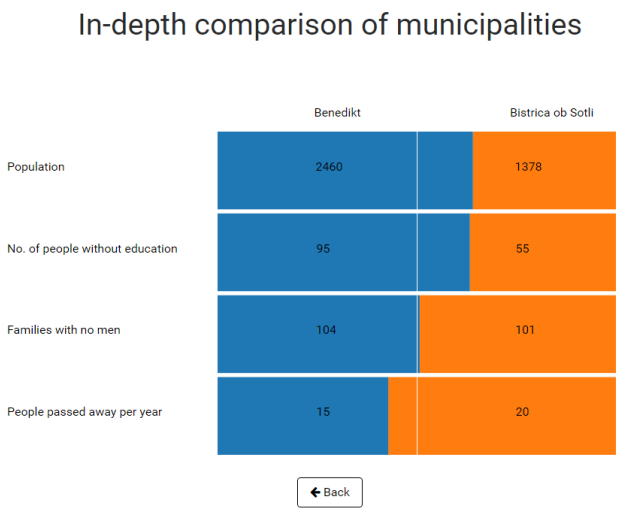


Slika 3.3: Primer linijskega grafa.

3.3.4 Implementacija izrisa paličnega diagrama

Za prikaz primerjave dveh občin smo izbrali palični diagram. Palčni diagram uporabniku predstavlja drugi nivo vizualizacije. Diagram se izrisuje kot prikaz razmerja podatkov, ki se sproži ob kliku na piko linijskega grafa. Začetek izrisa predstavlja metoda, ki pridobi podatke potrebne za izris takšnega diagrama. Ob klicu metode preverjamo, za kateri občini gre, leto, za katerega poizvedujemo, po katerih podatkovnih tipih poizvedujemo ter ali želimo normalizirane podatke. V tej metodi smo razvili algoritem, ki pridobljene podatke preuredi tako, da z njimi lahko izrišemo želeno vizualizacijo. Izris diagrama začnemo z določanjem velikosti celotne površine diagrama ter, glede na definirane odmike, določanjem velikosti za izris vizualizacije. Odmiki so prirejeni, tako da upoštevajo velikosti teksta, ki se bo izpisal za vsak izbran podatkovni tip. Metoda nato definira horizontalne ter vertikalne skale. Horizontalna skala za takšen tip diagrama določa prostor na pripadajoči, osi po kateri izrisujemo pravokotnike, ki predstavljajo razmerje vrednosti podatkov.

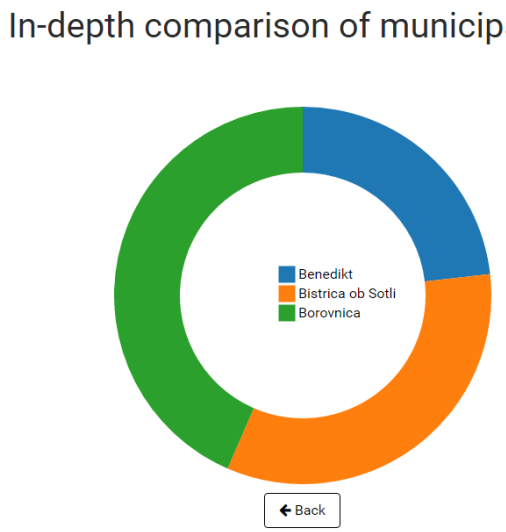
Vertikalna os določa višino pravokotnikov glede na število podatkovnih tipov, po katerih primerjamo. Po določenem ogrodju za vsako občino izrišemo pravokotnik, ki narašča po skali določeni na začetku metode. Tako po izrisu obeh občin dobimo vizualizacijo, ki učinkovito prikazuje razmerje podatkov. Po sredini diagrama metoda izriše belo črto, ki vidno prikazuje sredinsko vrednost. Metoda nad diagram pripne besedilo, z imenoma obeh občin, na levo stran pa naslove kriterijev. Diagram pred izrisom prekrijemo s kvadratom istih dimenzij kot je celotna risalna podlaga. Kvadrat se po izrisu krči od leve proti desni ter daje uporabniku občutek postopnega izrisa. Po končanem izrisu diagrama metoda podlogi pripne poslušalca dogodkov, ki je vezan na desni klik miškinega gumba. Ob proženju dogodka metoda prebere objekt, ki smo ji ga posredovali ob klicu ter vzpostavi prejšnje stanje. S tem dajemo uporabniku možnost vračanja nivo višje po izrisu grafov. Primer paličnega diagrama je viden na Sliki 3.4.



Slika 3.4: Primer izrisa paličnega diagrama.

3.3.5 Implementacija izrisa votlega tortnega diagrama

Tortni diagram je vizualizacija drugega nivoja. Prikaže se v primeru, da je uporabnik izbral en kriterij in več kot dve občini.
Najprej se določi velikost površine izrisa in odmikov samega diagrama od robov površine. Ker je diagram okrogle oblike, definiramo še radij, z definicijo širine kolobarja pa si zagotovimo potrebene podatke za ogrodje kasnejšega izrisa. Definicija osi tu ni potrebna, saj bomo diagram izrisali okoli sredine določene z ogrodjem. Določiti je potrebno le linearno skalo za barve. Okoli sredinske točke se definira ogrodje tortnega diagrama, kateremu nato na zunanji rob pripnemo pot s širino, ki smo jo definirali kot širino kolobarja. Pripenjanje poti se izvede postopoma, kar se na aplikaciji odraža kot animacija izrisa. Metoda v votlo sredino pripne po en kvadrat za vsako občino, ki je iste barve kot pot, ki predstavlja občino. Poleg kvadrata metoda doda še besedilo za povezovanje občine in določeno barvo. Metoda definira poslušalca dogodkov, ki ob uporabnikovem desnem kliku povrne stanje v prvi nivo izrisa in zažene animacijo izrisa. Primer takega grafa je viden na Sliki 3.5.

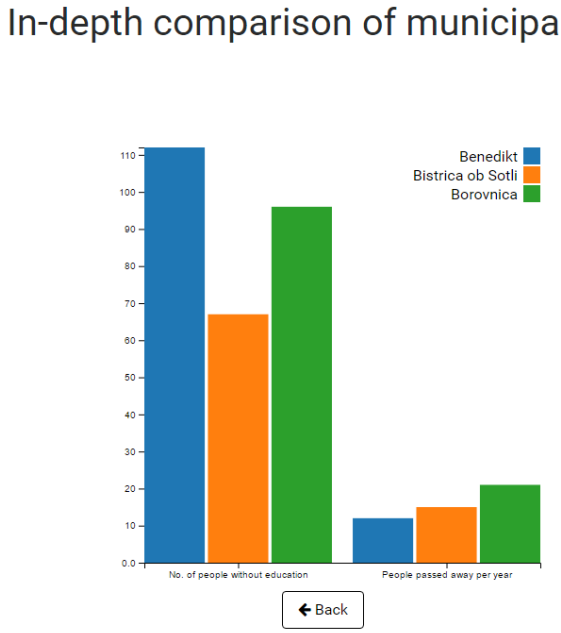


Slika 3.5: Primer prikaza votlega tornega grafa za štiri občine.

3.3.6 Implementacija izrisa gručnega stolpčnega grafa

Zadnji tip grafa, ki jih naša aplikacija lahko izriše je gručni stolpčni graf. Tudi ta graf je drugonivojski in se izrisuje, ko uporabnik izbere več kot dve občini in več kot en podatkovni tip. Poseben primer za uporabo takšnega izrisa je, kadar podatki pridobljeni za izris pokrivajo časovni interval ene enote. Linijski graf tako ne predstavlja najboljšega prikaza zato, že na prvem nivoju izrišemo gručni stolpčni graf.

Metoda je na začetku opremljena s podatki dimenzij risalne površine ter odmiki grafa od robov celotne površine. Za konsistentnost aplikacije definiramo skalo barv z enakimi podatki kot pri ostalih grafih. Sledijo definicije osi, ki so pri tem grafu malce drugačne. Horizontalna os vsebuje dvakratni izračun skale. Pri prvem izračunu metoda pridobi dolžino celotne osi, v katero umesti domeno glede na število kriterijev. Drugi izračun pa za vsak kriterij prvotno skalo razdeli še glede na število občin, za katere smo podali podatke. Tako definirane skale nam omogočajo, da ne glede na število podanih podatkov, vedno izrišemo enakomerno široke stolpce. Določimo tudi linearno skalo po vertikalni osi, ki bo odražala vrednost podatkov. Iz podatkov metoda z algoritmom razbere vrednosti kriterijev, katere pretvori na oznake skupin na horizontalni osi. Za nadaljnji izris grafa metoda najprej poskrbi za pretvorbo podatkov po skupinah in občinah. Ko ima metoda pretvorjene podatke, določi domeno vertikalne osi, saj za izris na enem grafu potrebujemo enotno skalo. Sledi izris pravokotnikov, ki za atribute višine, širine in pozicije dobijo podatke iz podane tabele ter ogrodja grafa. Glede na vrstni red izrisa metoda pravokotnikom določi barvo, isto barvo pa določi tudi kvadratom, ki jih pripne v del za legendo. Poleg teh kvadratov doda besedilo, ki uporabniku pove, katera barva predstavlja katero občino. Čez izris diagrama metoda doda pravokotnik enakih dimenzij kot celotna risalna podlaga. Slednji se zmanjšuje v smeri proti vrhu grafa ter tako daje občutek realnočasovnega izrisa diagrama. Na podlago diagrama ter posamične stolpce pripnemo poslušalca dogodkov, ki ob desnem kliku uporabnika povrne prejšnje stanje aplikacije. Primer gručnega stolpčnega grafa je viden na Sliki 3.6.



Slika 3.6: Primer gručnega stolpčnega diagrama.

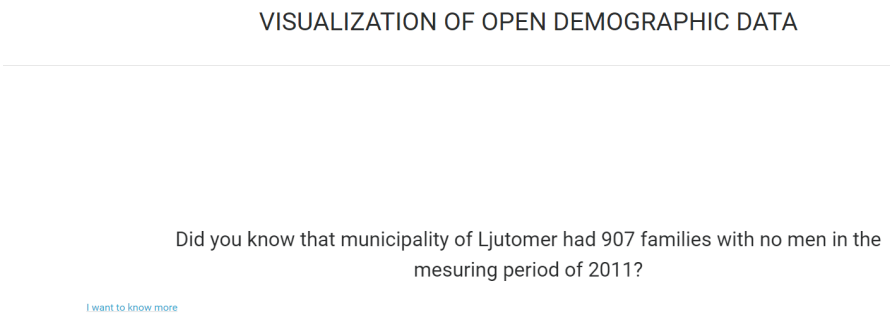
Poglavje 4

Uporabnikova pot

Namen aplikacije je uporabniku podati vizualno predstavitev podatkov, ki mu omogoča hiter pregled ter razumevanje le-teh. Pri razvoju smo upoštevali cilj diplomskega dela in željo, da uporabnika ne preobremenimo s podatki. Tako je nastala pot v treh korakih, ki uporabnika pripelje do želene vizualizacije.

4.1 Vstopna stran

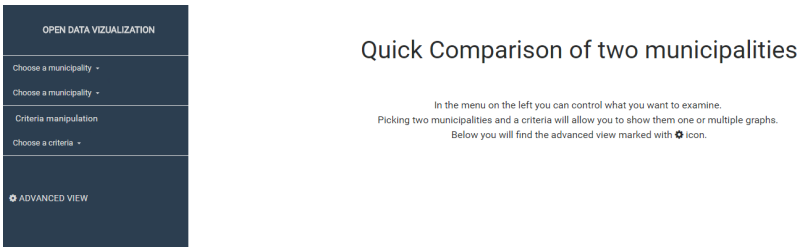
Vstopna stran je preprost izpis trivialnih podatkov, namenjenih, da pri uporabniku vzbudijo zanimanje ter s tem služijo kot uvod v aplikacijo samo. Primer je na Sliki 4.1.



Slika 4.1: Primer prikaza vstopnega pogleda, ki uporabniku ponudi naključno izbran podatek iz podatkovne baze.

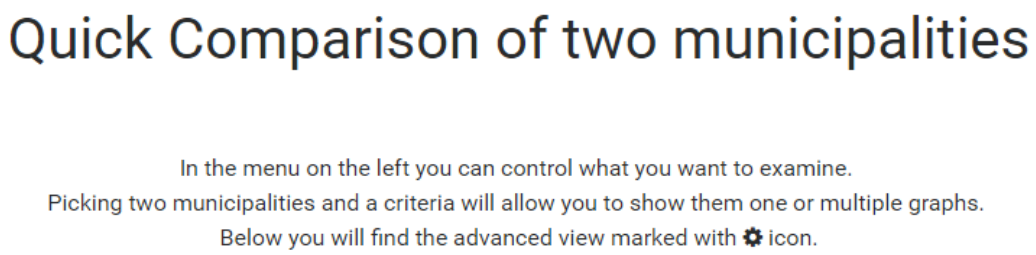
4.2 Druga stran - prva primerjava

Druga stran je namenjena spoznavanju z možnostmi in postavitvijo, ki jih aplikacija ponuja. Aplikacija ima na levi strani meni, kjer uporabnik s spustnih seznamov lahko izbere eno ali dve občini ter en kriterij. Če kriterija ne izbere, se izrišejo grafikoni z vsemi kriteriji, ki so na voljo. Ob vstopu na stran aplikacija v osrednjem delu prikaza uporabniku razloži uporabo menija. Primer navodil je viden na Sliki 4.2.



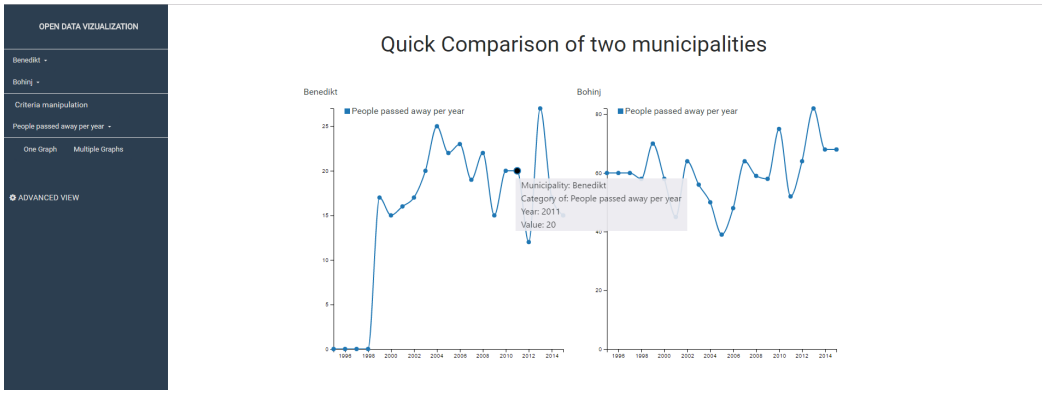
Slika 4.2: Primer navodil pri prvem vstopu na drugo stran aplikacije.

Podrobnejši ogled navodil je razviden na Sliki 4.3



Slika 4.3: Podrobnejši ogled navodil pri prvem vstopu na drugem pogledu aplikacije.

Po izbiri občin ter kriterija uporabniku drugi pogled aplikacije ponudi primerjavo po izbranem kriteriju. Primer je na Sliki 4.4.

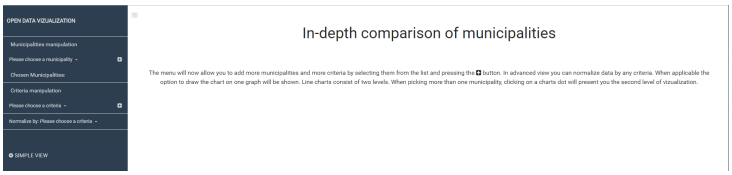


Slika 4.4: Primer osnovne primerjave med dvema občinama. Na sliki je viden izris ob uporabnikovi izbiri obeh občin, kriterija ter brez spreminjanja tipa izrisa. Ob postavitvi miškinega kazalca na eno izmed označenih točk grafa, se le-ta razširi, njena sredina pa se obarva črno. Poleg točke se uporabniku pokaže okvir, ki vsebuje dodatne podatke o izbrani točki.

Do glavnega dela aplikacije uporabnika povezuje gumb za podrobni pogled.

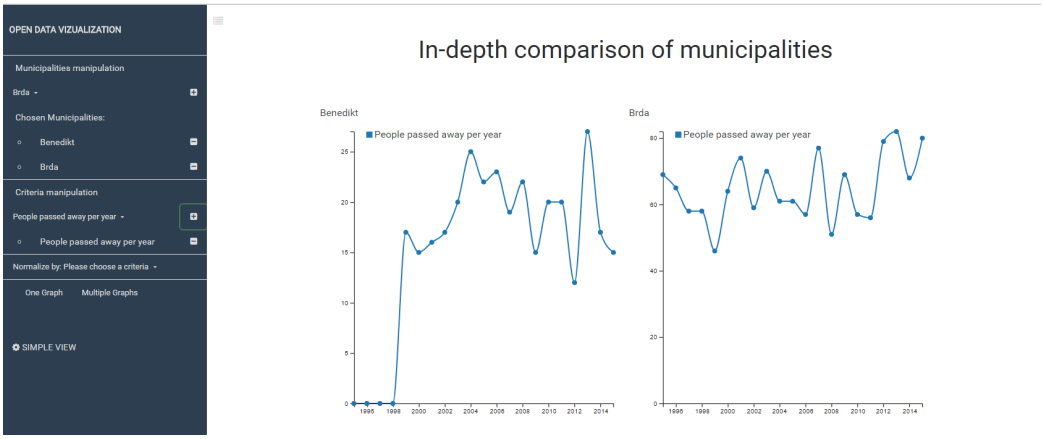
4.3 Tretja stran - glavni del aplikacije

Ob vstopu na tretjo stran uporabnika pričaka sporočilo o spremembi uporabniškega vmesnika na levi strani. Sporočilo predstavi nove možnosti pri uporabi vmesnika. Primer je viden na Sliki 4.5.



Slika 4.5: Primer navodil pri prvem vstopu na tretjo stran aplikacije in nove možnosti filtriranja podatkov, ki obsegajo poljubno število občin, poljubno število kriterijev ter normalizacijo po željenem kriteriju.

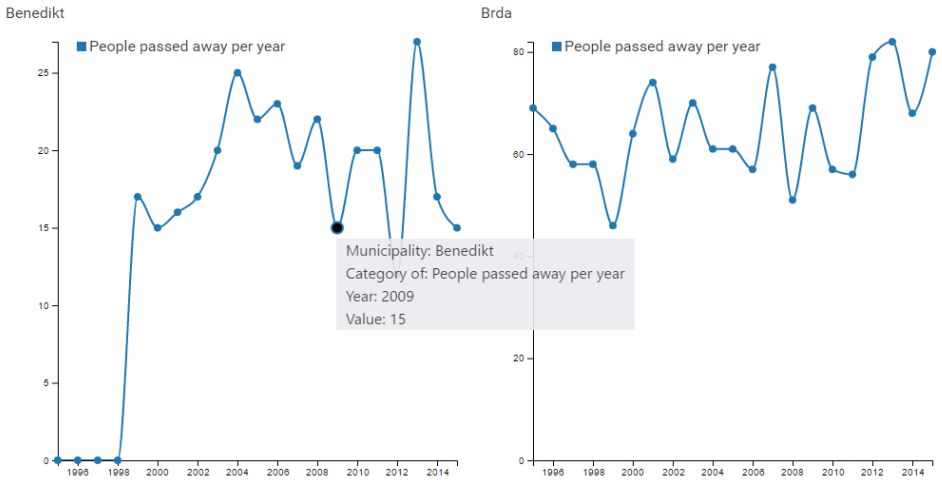
Uporabnik si po spoznavanju z novim menijem na levi strani izbere občine in tipe podatkov, po katerih želi primerjati. Osnovna primerjava med občinami je linijski graf. Primer je na Sliki 4.6.



Slika 4.6: Primer prikaza osnovnega linijskega grafa ob izbiri dveh občin ter enega kriterija. Način izrisa grafa ostaja nespremenjen, po en graf za vsako občino. Prav tako ni določena normalizacija podatkov.

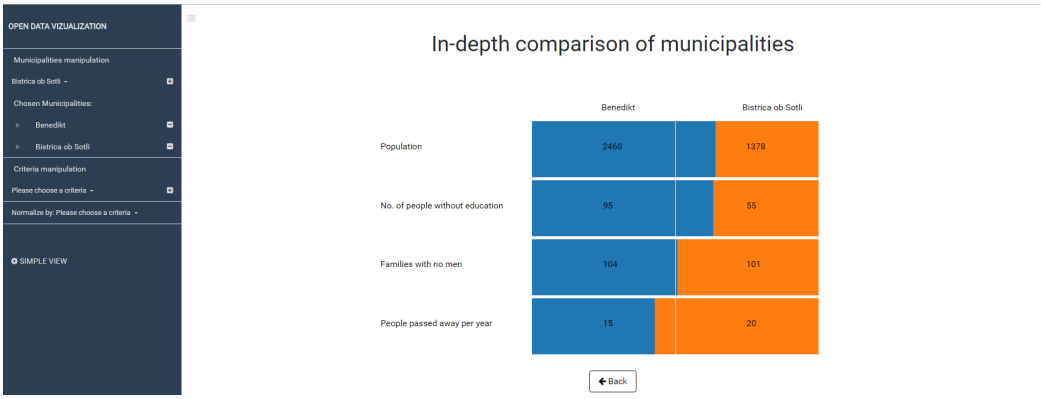
Skozi celotno aplikacijo vsaka pika na grafu predstavlja podatek, ki ga je aplikacija pridobila ob klicu na zaledni sistem. Ko uporabnik miškin kazalec

postavi na piko, se prikaže okno, ki predstavlja podatke. Primer je na Sliki 4.7.



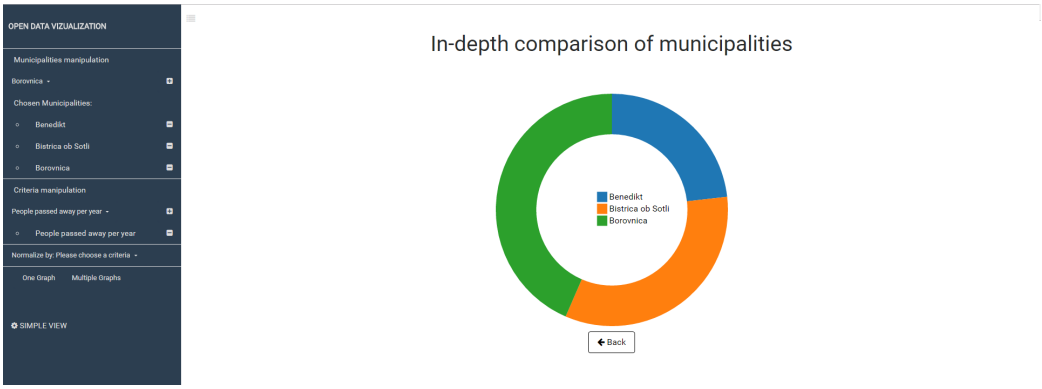
Slika 4.7: Primer prikaza podrobnosti podatkov na linijskem grafu.

Če želi uporabnik pridobiti podrobno primerjavo za trenutno izbrane kriterije ter občine, lahko klikne na piko na linijskem grafu. Glede na izbrane tipe podatkov in število občin, bo aplikacija predstavila razmerje na najbolj primeren način. Prehod na drugi nivo prikaza je možen ob izbiri vsaj dveh občin. Ko uporabnik izbere dve občini, aplikacija ob kliku na piko prikaže palični diagram z izbranimi kriteriji. Če uporabnik ne izbere nobenega tipa, aplikacija izriše vse podatkovne tipe. Primer je na Sliki 4.8.



Slika 4.8: Primer prikaza paličnega diagrama za vse kriterije, podatki normalizirani na tisoč prebivalcev. Nad paličnim diagramom sta imeni občin, ki ju podatki predstavljajo, sredinska vrednost je prikazana z belo črto preko barvnih prikazov vrednosti. Kriteriji so imenovani na levi strani diagrama.

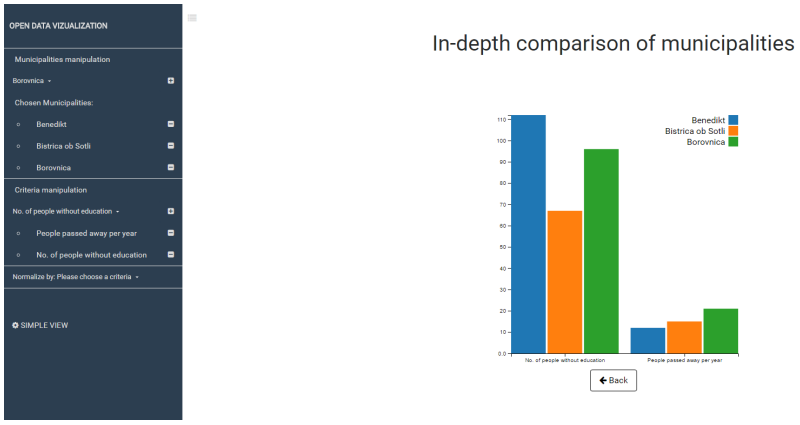
Kombinacija klika na piko, treh ali več občin in enega kriterija bo uporabniku prikazala votli tortni diagram za prikaz razmerja podatkov. Primer je viden na Sliki 4.9.



Slika 4.9: Prikaz votlega tortnega diagrama. Za izkoristek prostora smo legendo postavili v votli del diagrama. Vsak barvni del predstavlja količino posamezne občine v prikazu razmerja.

Ob izbiri več kot dveh občin in več kot dveh kriterijev, bo aplikacija uporabniku ponudila pogled na gručni stolpčni diagram, kot je prikazano na

Sliki 4.10.



Slika 4.10: Prikaz gručnega stolpčnega diagrama. Prikaz predstavlja izbiro dveh kriterijev s po tremi občinami. Izbrana ni nobena normalizacija podatkov.

Poglavje 5

Sklepne ugotovitve

Izdelava takšne aplikacije je smotrna zaradi številnih razlogov. V prvi vrsti rešuje problem tabelarnega prikaza podatkov, ki ga nudi država. Aplikacija uporabniku omogoča manipulacijo podatkov po poljubnih kriterijih, kar ji daje prednost pred ostalimi aplikacijami na trgu. Večina aplikacij ponuja omejeno mero prilagoditve prikaza podatkov, kar smo nadgradili do stopnje, kjer lahko uporabnik popolnoma nadzira vizualizacijo podatkov. Prilagodljivost podatkovnih filtrov daje aplikaciji zmožnost adaptacije na različna področja. Aplikacijo bi tako lahko uporabili v vseh panogah, kjer poteka obdelava podatkov. Uporabili bi jo lahko na primer v bančništvu za pregled poslovanja, pri trgovanju na borzi za pregled nihanja vrednosti delnic ali pa v farmaciji za pregled učinkov zdravil. Prav zaradi svoje sposobnosti prikaza kakršne koli kombinacije podatkov je aplikacija zanimiva tudi za laične uporabnike, ter kot taka pripomore k osveščanju populacije o odprtih podatkih. Dodano vrednost naši aplikaciji predstavljajo tudi izbire grafov. Grafi, ki so bili oblikovani glede na dobre prakse vizualizacije, zagotavljajo konsistenten ter nezavajajoč prikaz podatkov.

Za dosego cilja smo razvili dva glavna dela aplikacije, strežniški ter uporabniški del. Na strežniškem delu aplikacije smo vzpostavili podatkovno bazo, katero lahko polnimo z uporabo skripte, ki smo jo razvili za uvoz podatkov. Skripta omogoča generični uvoz podatkov ter tako razširja uporabnost

aplikacije na številne trge. Strežniški del aplikacije skrbi tudi za pridobivanje podatkov iz podatkovne baze ter prvotno manipulacijo le-teh. Pridobljene podatke posredujemo drugemu glavnemu delu, uporabniški aplikaciji. Uporabniški del aplikacije skrbi za prikaz podatkov na način, ki je uporabniku prijazen. To pomeni, da smo pri razvoju uporabniškega vmesnika pazili na uporabniško izkušnjo. Vmesnik tako ni prezahteven, saj v stopnjah pripelje uporabnika do končne oblike ter mu na poti razloži, kako ga uporabljati. Razlaga ter tako intuitivnost vmesnika je zagotovljena s pojavnimi sporočili, kjer pa smo pazili, da le-teh ni preveč saj hitro preidejo iz pomoči v nadlogo. Ko uporabnik razume, kako upravljati z aplikacijo, lahko primerja raznovrstne podatke na različne načine, kar mu pomaga do vzpostavitve za njega potrebnih informacij. Prijaznost do uporabnika, razumljivost vmesnika ter posledično njegovo učinkovitost smo zagotovili tako, da smo sledili principom, ki smo jih opisali v podpoglavju Oblikovanje uporabniškega vmesnika.

Zasnova aplikacije ponuja možnosti za nadaljnji razvoj aplikacije. Dinamična zgradba aplikacije ter izbira tipa podatkovne baze omogočata skrbniku preprosto razširitev zbirke podatkov do izrednih razsežnosti. Aplikacijo bi se lahko razširilo kot odprto orodje, kjer bi vsak uporabnik lahko dodajal podatke v podatkovno bazo. Možnosti razširitev pa so na primer še izvoz primerjav podatkov ter prikaza podatkov na zemljevidu.

Literatura

[1] Angular2 dokumentacija Dosegljivo: *https://angular.io/.* [Online; Dostopano 3.1.2017].

[2] Bootstrap dokumentacija Dosegljivo: *http://getbootstrap.com/.* [Online; Dostopano 3.1.2017].

[3] Brian Carter, HTML Architecture, a Novel Development System (HANDS): An Approach for Web Development, *Information and Computer Technology (GOCICT), 2014 Annual Global Online Conference*, str. 90 - 95, 2014, 978-1-4799-8311-7

[4] Pablo Navarro Castillo, Mastering D3.js. *Packt Publishing Ltd.*, 2014, 978-1-78328-627-0

[5] Yu Chen, Research on Optimized Design of Kansei Engineering-based Web Interface, *2013 International Conference on Computational and Information Sciences* ,978-0-7695-5004-6/13

[6] Direktiva o spremembi Direktive o ponovni uporabi informacij javnega sektorja, "Direktiva 2013/37/EU Evropskega parlamenta in sveta z dne 26. junija 2013", Uradni list Evropske unije L.175/1, 27.6.2013

[7] Informacije javnega značaja Dosegljivo: *http://www.upravneenote.gov.si/informacije_javnega_znacaja/kaj_so_informacije_javnega_znacaja/.* [Online; Dostopano 3.1.2017].

[8] Chaokui Li, Wu Yang , The distributed storage strategy research of remote sensing image based on Mongo DB , *Earth Observation and Remote Sensing Applications (EORSA), 2014 3rd International Workshop*, 2014, 978-1-4799-4184-1

[9] MongoDB Dosegljivo: *https://www.mongodb.com/*, [Online; Dostopano 3.1.2017].

[10] Node.js dokumentacija Dosegljivo: *https://nodejs.org/en/.* [Online; Dostopano 3.1.2017].

[11] npm dokumentacija Dosegljivo: *https://www.npmjs.com/.* [Online; Dostopano 3.1.2017].

[12] Edward R. Tufte. The Visual Display of Quantitive Information. Cheshire : Graphics Press, cop. 2001

[13] TypeScript dokumentacija. Dosegljivo: *http://www.typescriptlang.org/.* [Online; Dostopano 3.1.2017].

[14] Gerard Wagner, "Introduction to simulation using JavaScript", v zborniku *2016 Winter Simulation Conference (WSC)*, str. 148 - 162, 2016

[15] W3Schools Dosegljivo: *http://www.w3schools.com/html/html_intro.asp.* [Online; Dostopano 3.1.2017].

[16] Zakon o dostopu do informacij javnega značaja Dosegljivo:*http://www.pisrs.si/Pis.web/pregledPredpisa?id=ZAKO3336.* [Online; Dostopano 3.1.2017].